Universidad Carlos III de Madrid

Escuela Politécnica Superior

Departamento de Ingeniería de Sistemas y Automática

DISEÑO Y DESARROLLO DEL INTERFAZ PARA LA TELEOPERACIÓN DE UN BRAZO ROBÓTICO

PROYECTO FIN DE CARRERA

INGENIERIA INDUSTRIAL

Autor: Laura Navío Haro

Tutor: José María Armingol Moreno

Madrid 2013

AGRADECIMIENTOS

Quisiera agradecerle a mis padres y hermano el apoyo que me han brindado durante todo este tiempo, siempre han estado a mi lado ayudándome en todo lo posible para llegar aquí y han sabido tratarme en cada momento, sobre todo en los más difíciles y complicados.

A Martín, por apoyarme y animarme para finalizar este proyecto, que ha puesto todo de su lado para que hoy esté escribiendo estas líneas, y no desespere con el *mañana*.

También quiero agradecerles su ayuda a José Luis Muñoz Lozano y a Carlos Alberto Díaz Hernández, por la atención y ayuda que me han prestado durante todo el desarrollo del proyecto.

Y en especial a José María Armingol, que una vez más ha confiado en mí para realizar el proyecto final de carrera, me ha dado libertad para realizarlo, y a pesar de las últimas prisas ha seguido estando ahí.

ÍNDICE

ÍNDICE DE ILUSTRACIONES Y TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1 INTRODUCCIÓN Y OBJETIVOS

El proyecto que nos ocupa se inició al cursar varias asignaturas relacionadas con la robótica industrial, tras superarlas surgió la posibilidad de realizar el proyecto final de carrera con un robot articulado de 6 grados de libertad para fines formativos, la idea era desarrollar una interfaz con la que los alumnos pudieran manipular el robot, y que con el tiempo, otros estudiantes de ingeniería fueran ampliándola para crear una aplicación completa de teleoperación para este brazo articulado.

La propuesta era interesante, anteriormente ya había participado en un proyecto en conjunto de diseño y manipulación de un microrrobot para un concurso estudiantil internacional. Esta vez los retos iban más allá, desarrollar el programa de manipulación de un robot comercial de forma individual.

Por ello, los objetivos de este proyecto se fundamentan en el autoaprendizaje de nuevas herramientas de desarrollo de software, nuevos lenguajes de programación y no menos importante, afrontar este trabajo en solitario.

1.2 ESTRUCTURA DE LA MEMORIA

El contenido de esta memoria se distribuye en distintos capítulos en función de los bloques temáticos que se han desarrollado a lo largo de este proyecto, los cuales son:

ESTADO DEL ARTE

Este proyecto se desarrolla en el ámbito de la robótica, es por ello que se precisa exponer una introducción de este amplio campo de conocimiento, comentando sus aplicaciones a todos los niveles: práctico y funcional, industrial, investigación, desarrollo de nuevas aplicaciones y formativo. Este último aspecto es la guía de este proyecto, la formación no sólo en la teleoperación de robots industriales, sino en la formación en otras materias como las telecomunicaciones o la informática, pues para llevar a cabo no sólo han sido necesarios los

conocimientos de robótica y automática acerca de la cinemática de los mecanismos y el estudio de su comportamiento, sino que ha sido imprescindible aprender otras materias y desarrollar nuevas habilidades; como nuevos lenguajes de programación y comunicaciones entre dispositivos, o el manejo de nuevas herramientas de desarrollo.

ENTORNO HARDWARE

En este apartado se detallan todos los componentes hardware principales con los que se ha trabajado para desarrollar este proyecto. Desde los aspectos generales del robot hasta el detalle de cada módulo y elemento de unión, así como el circuito eléctrico y accionamiento de parada de emergencia, vital en la teleoperación de este tipo de robots. También se hace un breve resumen de la teoría cinemática empleada basada en los estudios desarrollados por Denavit y Hartenberg.

ENTORNO SOFTWARE

Este capítulo recoge una breve descripción del soporte software que se ha utilizado, desde el sistema operativo empleado hasta el entorno de desarrollo y el lenguaje de programación. Este apartado es muy importante porque este robot ofrece diferentes alternativas de todos estos aspectos, y se hace necesario exponer el porqué de las decisiones tomadas acerca de estos puntos.

COMUNICACIONES

De nada sirve tener todo un dispositivo hardware y un desarrollo software, si no se es capaz de comunicar ambos. Este capítulo detalla el tipo de comunicación existente, las características de la misma así como el proceso tratamiento de la información y envío de datos.

EL PROBLEMA Y LA SOLUCIÓN. INTERFAZ GRÁFICA

En este apartado se realiza un análisis de los requerimientos para desarrollar este proyecto en base los objetivos fijados en un principio, y las decisiones tomadas para alcanzarlos. Se detalla el funcionamiento de la aplicación desarrollada haciendo una partición funcional, y explicando cuidadosamente el comportamiento de la interfaz gráfica que se ha creado.

CONCLUSIONES Y MEJORAS

Este capítulo se dedica a las conclusiones y mejoras que se pueden realizar a este proyecto. Se exponen cada uno de los resultados obtenidos y se ofrecen ideas o sugerencias para la ampliación de esta interfaz de cara al futuro.

BIBLIOGRAFÍA

Se incluye el material de referencia y el material de consulta, así como las diferentes fuentes de Internet que se han consultado para la realización de este proyecto

ANEXOS

En este apartado último se incluyen los anexos; información de interés tal como las funciones de la librearía principal del robot m5apiw32.h, ejemplo de un informe de chequeo generado, las hojas de características así como otros extractos de los manuales de funcionamiento de Robonik.

CAPÍTULO 2

ESTADO DEL ARTE

La necesidad de mejorar las tareas diarias, es lo que lleva a una sociedad a un progresivo desarrollo tecnológico. Así, desde un comienzo en la agricultura o la caza, hasta el transporte de mercancías y personas pasando por la guerra y la conquista de territorios.

De esta forma, en la Historia podemos diferenciar distintas edades, las cuales diferenciamos por los útiles de los que disponía el ser humano para realizar sus tareas rutinarias, éstos como puede comprobarse, dependen del conocimiento sobre los diferentes aspectos científico-tecnológicos que generación tras generación se van transmitiendo y gracias a los cuales han surgido numerosas ramas de conocimiento como son la electricidad, electrónica, mecánica, química, informática y telecomunicaciones.

Ilustración 2.1: Evolución tecnológica.

Con este fin; mejorar la vida diaria en cuanto a facilidades domésticas y de trabajo se refiere, el ser humano ha inventado infinidad de máquinas que mejoran los procesos manuales, siendo muchas las que han sustituido a las propias personas; a este tipo de máquinas se denominan ***robots***, y han dado lugar a la rama científica de la Robótica.

Según la Real Academia Española, la Robótica es: *"La Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales."* [1]

Así mismo, podemos encontrar otras definiciones de Robótica:

"La robótica es la ciencia encaminada a diseñar y construir aparatos y sistemas capaces de realizar tareas propias de un ser humano." [2]

"El diseño, fabricación y utilización de máquinas automáticas programables con el fin de realizar tareas repetitivas como el ensamble de automóviles, aparatos, etc. y otras actividades. Básicamente, la robótica se ocupa de todo lo concerniente a los robots, lo cual incluye el control de motores, mecanismos automáticos neumáticos, sensores, sistemas de cómputos, etc." [3]

"La robótica es un campo tecnológico que engloba varias disciplinas, como son mecánica, informática, automática, electricidad, electrónica y organización." [4]

En la actualidad el desarrollo de estas tecnologías es muy avanzado, y en los últimos tiempos hemos podido comprobar la rápida y eficaz evolución de las mismas gracias a la competencia en el sector industrial. De esta forma no se puede entender el gran avance tecnológico de nuestra era sin la participación de los robots, que como se ha indicado antes, incluso han llegado a sustituir al ser humano en numerosas tareas; bien trabajos peligrosos, de gran esfuerzo físico, repetitivos, alto nivel de procesado de datos o de precisión, así se destinan tanto a procesos domésticos, como de vigilancia, investigación, medicina, militares, etc.

La robótica se basa en aspectos fundamentales como las matemáticas, electrónica, informática, mecánica e inteligencia artificial, los cuales están tan desarrollados a día de hoy, que los costes de producción, la simplificación de los sistemas y la reducción de su tamaño es sencilla.

En la vida cotidiana podemos destacar los teléfonos móviles y los ordenadores como ejemplo de rápida y sorprendente evolución. A pesar que en apariencia parece que todo está inventado en este campo, cada día los profesionales nos sorprenden con nuevas características, funcionalidades, aplicaciones y tecnologías aplicadas a estos objetos comunes. Resulta extraño encontrar un hogar de clase media en los países desarrollados que no tenga un ordenador o un teléfono móvil, ya que han reducido tanto su coste, que están al alcance de la mayoría de los bolsillos.

Ilustración 2.2: Evolución del teléfono móvil.

Ilustración 2.3: Evolución de los iMac.

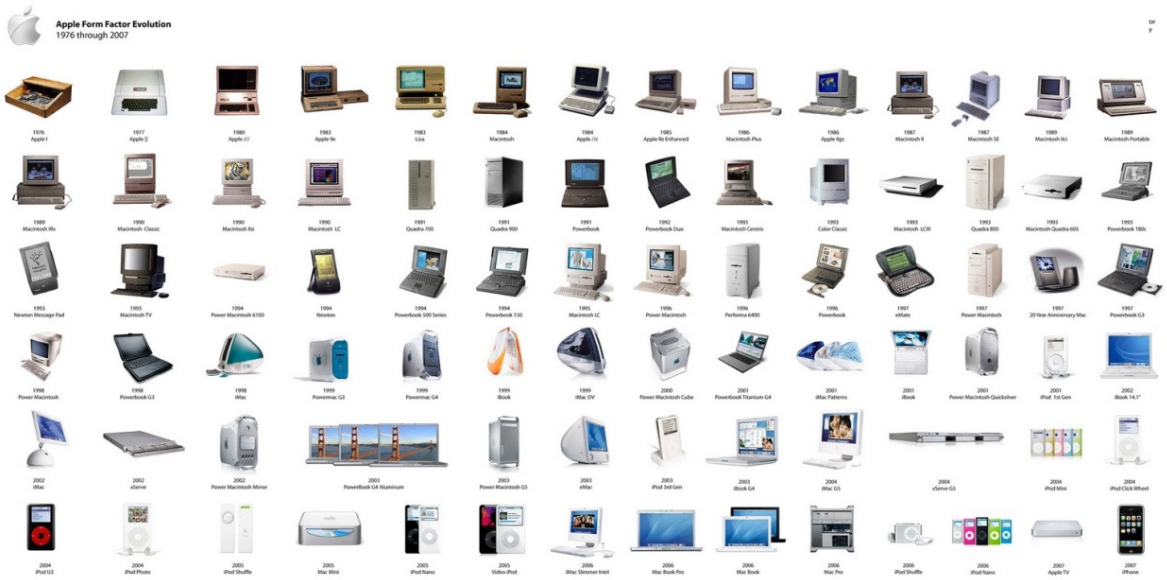


Ilustración 2.4: Evolución de los ordenadores Apple.

Aunque es muy difícil hablar del final de esta espectacular evolución, si podemos remontarnos a sus orígenes para comprender mejor cual ha sido la vía de desarrollo de la tecnología; su motivación y su funcionalidad. En sus inicios se encontraba la necesidad de aumentar la productividad a la vez que la calidad, de tal forma que desde el inicio del siglo XX el empleo de máquinas y robots se realizó en la fabricación en masa de un mismo producto, lo que contrasta drásticamente con la necesidad actual, donde se busca fabricar productos exclusivos, casi personalizados.

La historia de la robótica se antoja corta pero intensa, así, en los siguientes apartados se realizará un resumen de ésta, destacando los puntos clave desde los primeros autómatas hasta los robots actuales, haciendo hincapié en la robótica modular, rama en la que se desarrolla este proyecto.

2.1 ANTECEDENTES HISTÓRICOS

En primer lugar es necesario conocer a qué nos referimos con el término "autómata", del griego automatos (αὐτόματος) que significa espontáneo o con movimiento propio, para así poder entender la historia de la robótica, encontramos diferentes definiciones tales como [5];

1. Máquina que contiene un mecanismo que le permite realizar determinados movimientos.
2. Máquina que imita la figura y los movimientos de un ser animado.
3. Dispositivo o conjunto de reglas que realizan un encadenamiento automático y continuo de operaciones capaces de procesar una información de entrada para producir otra de salida.
4. Persona que se deja dirigir o actúa condicionada y maquinalmente.

Otra definición similar nos dice que un autómata es [6];

1. Instrumento o aparato que encierra dentro de sí el mecanismo que le imprime determinados movimientos.
2. Máquina que imita la figura y los movimientos de un ser animado.
3. Persona estúpida o excesivamente débil, que se deja dirigir por otra.

Una vez conocemos el significado del término "autómata", se puede decir que desde sus inicios, el ser humano a anhelado la creación de vida artificial, lo cual consiguió con los primeros autómatas de la historia, que de forma más o menos sencilla acertaron a imitar los movimientos y sonidos de animales y elementos naturales con el fin de entretener a los dueños de estos artefactos. Otro uso fue el religioso, empleados en templos y estatuas causaron respeto y admiración entre los adoradores. Los chinos y los griegos destacaron en este ámbito lúdico, en general sus autómatas eran ingenios mecánicos de cierta complejidad que desarrollaban un programa fijo y que no empleaban necesariamente la noción de realimentación. Un aspecto fundamental es la repetición, con el tiempo los hombres se dieron cuenta de que existían actividades repetitivas en su vida cotidiana en las que podían sustituir su trabajo manual por unos artilugios que con un complejo sistema fueran capaces de repetir las mismas labores que realizaban, de esta forma, surgieron las máquinas. Los grandes precursores de este tipo de máquinas fueron los árabes, gracias a sus conocimientos matemáticos adquirieron gran precisión en la construcción de estos artefactos, siendo uno de sus grandes inventos, el reloj mecánico.

Los primeros autómatas datan del año 1500 a. C., Amenhotep, hermano de Hapu, construye una estatua de Memon, el rey de Etiopía, que emite sonidos cuando la iluminan los rayos del sol al amanecer. King-su Tse, en China, en el 500 a. C. inventa una urraca voladora de madera y bambú y un caballo de madera que salta. Entre el 400 y 397 a. C., Archytar de Tarento (inventor del tornillo y la polea) construye un pichón de madera suspendido de un pivote, el cual rotaba haciendo de surtidor de agua o vapor, simulando el vuelo. En el año 206 a. C., fue encontrado por el primer emperador Han el tesoro de Chin Shih Hueng Ti consistente en una orquesta mecánica de muñecos.

Como autómatas prácticos de estos inicios se pueden destacar dos inventos de Cresibio datados entre el 300 y 270 a. C., como son la clepsidra (reloj de agua) y un órgano que funciona también con agua, o la catapulta repetitiva de Filon de Bizancio entre el 220 y 200 a. C.

Situados ya en la época después de Cristo, podemos incluso encontrar uno de los primeros libros que detallan algunos de estos dispositivos, *"Autómata"*, data del año 62 y fue escrito por Herón de Alejandría. Entre los objetos descritos se encuentran distintas aves que vuelan, gorjean y beben que ellos fueron diseñadas como juguetes, sin mayor interés por encontrarles aplicación, conformando así su teatro automático en el cual las figuras que se encuentran montadas en una caja cambian de posición ante los ojos de los espectadores: pájaros cantores, trompetas que suenan, medidores de la fuerza del vapor, animales que beben, termoscopios, sifones y máquinas que operaban con monedas. Otro de los autómatas bien conocidos era "Hércules y el dragón", reconstrucción de Giovanni Battista Aleoti 1589, cuando Hércules golpeaba la cabeza del dragón éste arrojaba agua sobre su cara. Ambos pueden verse en la Ilustración 2.5. También recordar "The automaton theatre", que representa la historia de la guerra de Troya.

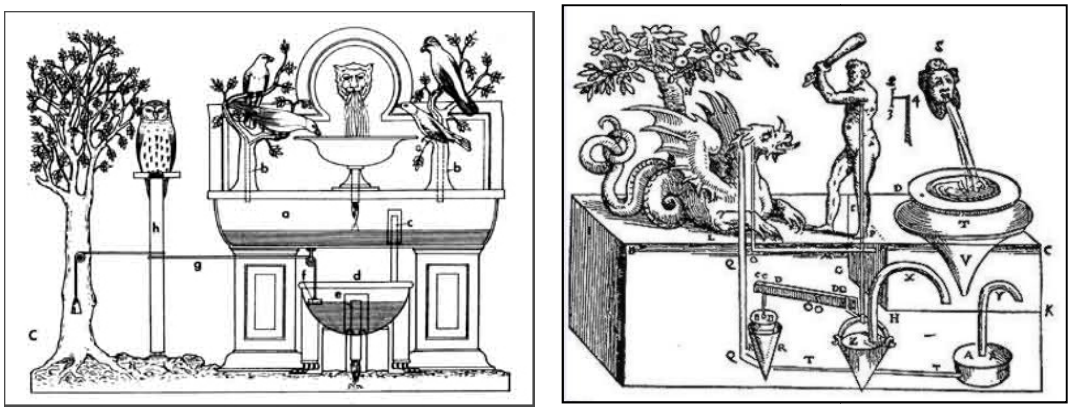


Ilustración 2.5: Pájaros de Herón y "Hércules y el dragón".

Sin embargo, también describe algunos objetos con aplicación práctica como un molino de viento para accionar un órgano o un precursor de la turbina de vapor.

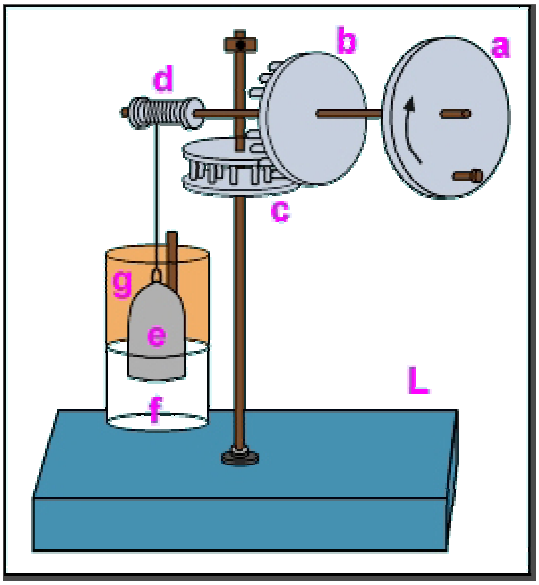


Ilustración 2.6: Caja mágica de Herón.

Por esta época también se diseñan ingeniosos mecanismos como la máquina de fuego que abría puertas de los templos o altares mágicos donde las figuras apagaban el fuego de la llama.

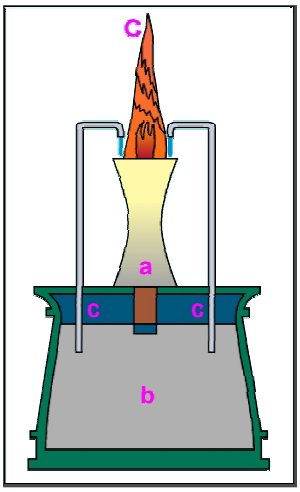


Ilustración 2.7: Altar mágico.

En Roma existía la costumbre de hacer funcionar juguetes automáticos para deleitar a los huéspedes. Trimalco ofreció en su famoso banquete, pasteles y frutas que arrojaban un chorro de perfume cuando se hacía una ligera presión sobre un príapo de pasta, en cuyo regazo estaban colocados dichos manjares.

Sin embargo no todos estos inventos se desarrollan en la cuna de la cultura clásica, en tierras orientales también hubo una gran evolución; del año 335 d. C. data un carro de cuatro ruedas construido en madera de sándalo con una figura de buda en la parte superior fabricado por Hsieh Fec. Siglos más tarde Huang Kun construyó barcos con figuras de animales, cantantes, músicos y danzarines que se movían (700 d. C.), mientras que Yang Wu-Lien construye un mono que extiende sus manos y dice "¡Limosna! ¡Limosna!", guardando su recaudación en una bolsa cuando alcanza un peso determinado (770 d. C.). Posteriormente, en el año 840 el príncipe Kaya, hijo del Emperador Kannu, construye una muñeca que derrama agua, y en el 890, Han Chih Ho hace un gato de madera que caza ratas y moscas que bailan.

No podemos olvidar mencionar otra gran cultura sin la cual, no entenderíamos la automática como hoy la conocemos; la cultura árabe, ésta heredó y difundió los conocimientos de los griegos, utilizándolos no sólo para realizar mecanismos destinados a la diversión, sino que les dieron una aplicación práctica, introduciéndolos en la vida cotidiana de la realeza. Ejemplos de estos son diversos sistemas dispensadores automáticos de agua para beber o lavarse, o el reloj que en el año 809 regaló a Carlomagno el califa Harún Al-Raschid, en el cual aparecían figuras que daban la hora. Mucho más tarde Al-Jazari (1260), uno de los grandes ingenieros de su época, inventor del cigüeñal y de los primeros relojes mecánicos movidos por pesos y agua, escribió *"El libro del conocimiento de los ingeniosos mecanismos"*, considerada una de las obras imprescindibles sobre historia de la tecnología.

También de este período son otros autómatas de los que hasta nuestros días sólo han llegado referencias no suficientemente documentadas, como el hombre de hierro de Alberto Magno (1204-1282) cuya leyenda cuenta que era un especie de mayordomo de hierro, cristal y cuero capaz de andar, abrir la puerta y saludar a los visitantes, o la cabeza parlante de Roger Bacon (1214-1294), hecha de latón y que podía responder a preguntas sobre el futuro, también podemos destacar la que fabricó Alberto Magno con forma de mujer, la cabeza parlante del Papa Silvestre II que respondía aleatoriamente "si" o "no". En el año 1235, Villard d'Honnecourt escribe un libro con bocetos que incluyen secciones de dispositivos mecánicos, como un ángel autómata, e indicaciones para la construcción de figuras humanas y animales.

Otro ejemplo relevante de la época fue el Gallo de Estrasburgo, véase Ilustración 2.8, funcionó desde 1352 hasta 1789, este es el autómata más antiguo que se conserva en la actualidad, formaba parte del reloj de la catedral de Estrasburgo, al dar las horas movía el pico y las alas tres veces.

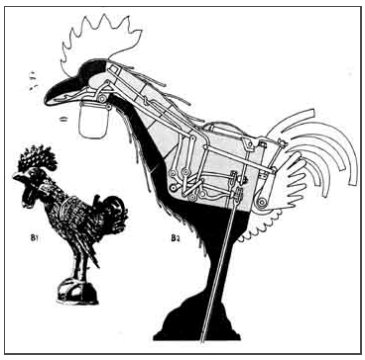


Ilustración 2.8: Gallo de Estrasburgo.

Durante los siglos XV y XVI algunos de los más relevantes representantes del renacimiento se interesan también por los ingenios descritos y desarrollados por los griegos. Es conocido el León Mecánico construido por Leonardo Da Vinci (1452-1519) para el rey Luis XII de Francia, que se abría el pecho con su garra y mostraba el escudo de armas del rey.

Ilustración 2.9: León de Leonardo da Vinci.

En España es conocido el hombre de palo construido por Juanelo Turriano (Giovanni Torriani) en el siglo XVI para el emperador Carlos V. Este autómata con forma de monje, andaba y movía la cabeza, ojos, boca y brazos. En 1558, el duque de Baviera hizo construir en

Nüremberg una casa de muñecas. Hacia finales del siglo XVI el mecánico alemán Cristóforo Schissler fabricó una muñeca automática que se movía con gran desenvoltura. A pesar de esto la edad de oro es el siglo XVIII, siglo mecánico por excelencia.

Durante los siglos XVII y XVIII se crearon ingenios mecánicos los cuales se pueden considerar el germen de los robots actuales ya que por aquel entonces mostraban características comunes a los de hoy en día. Estos dispositivos fueron creados en su mayoría por artesanos del gremio de la relojería. Su misión principal era la de entretener a las gentes de la corte y servir de atracción a las ferias. Dichos autómatas representaban figuras humanas, animales o pueblos enteros. Así, en 1610 el relojero Aquiles Langenbucher fabricó instrumentos musicales que sonaban solos, más tarde en 1649, cuando Luis XIV era niño, un artesano llamado Camus (1576-1626) construyó para él un coche en miniatura con sus caballos, sus lacayos y una dama dentro y todas las figuras se podían mover perfectamente. Salomón de Camus también construyó fuentes ornamentales y jardines placenteros, pájaros cantarines e imitaciones de los efectos de la naturaleza.

Según P. Labat, el general de Gennes construyó en 1688 un pavo real que caminaba y comía. Este ingenio pudo servir de inspiración a Jacques de Vaucanson (1709-1782) que no contento con haber realizado un modelo de telar mecánico al que debe su sólida fama, en 1738 expuso en Paris una serie de autómatas entre los cuales había un músico de flauta de tamaño natural, capaz de ejecutar doce piezas distintas con gran naturalidad (el ingenio consistía en un complejo mecanismo de aire que causaba el movimiento de dedos y labios, como el funcionamiento normal de una flauta); sin embargo, su mejor trabajo fue un "ánade", el cual puede observarse en la Ilustración 2.10. Según Sir David Brewster en un escrito de 1868, describe este pato diciendo que es "*la pieza mecánica más maravillosa que se haya hecho*". El pato alargaba su cuello para tomar el grano de la mano y luego lo tragaba y lo digería. Podía beber, chapotear y graznar, también imitaba los gestos que hace un pato cuando traga con precipitación. Los alimentos los digería por disolución y se conducía por unos tubos hacia el ano, donde había un esfínter que permitía evacuarlos. Aunque Vaucanson murió en 1782, todavía en 1805 Goethe pudo admirar el ánade ya estropeada, pero aún capaz de comer. Además, destacar que por instigación de Luis XV, intentó construir un modelo con corazón, venas y arterias, pero murió antes de poder terminar esta tarea.

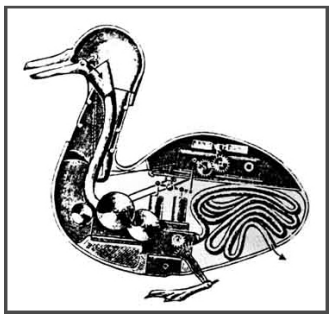


Ilustración 2.10: Pato de Vaucanson.

En el siglo XVIII distintos artistas mejoraron las cabezas parlantes creadas por Alberto Magno y Roger Bacon, estos dispositivos conseguían la "voz" a través de distintos sistemas,

siendo el primero que lo documentó Christian Gottlieb Kratzenstein (1723-1795), lo realizaba con un sistema de tubos de órgano con los que reproducía las vocales. Más tarde Wolfrang von Kempelen (1734-1804) documentaba su sistema a modo de fuelle con el que se controlaba el aire modulando sonidos gracias al cual se conseguía la reproducción de frases. Así, en el siglo XIX Joseph Faber fabricó la mejor máquina de este tipo hasta el momento; Euphonia, era capaz de recitar el alfabeto, responder preguntas, susurrar o reir.

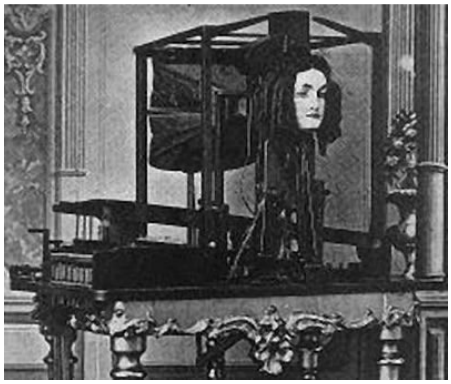


Ilustración 2.11: Cabeza parlante de Joseph Faber.

No podemos abandonar este siglo sin nombrar a Robert Houdini, el cual construye una muñeca que escribe, así como un pastelero, un acróbata, una bailarina en la cuerda floja, un hombre que apunta con una escopeta y una artista del trapecio. Mientras que Thomas Alva Edison construyó en el año 1891 una muñeca que hablaba.

Famoso también fue el relojero suizo Pierre Jaquet Droz (1721-1790) y sus hijos Henri-Louis y Jaquet pues construyeron diversos muñecos capaces de escribir (1770), dibujar (1772) y tocar diversas melodías en un órgano (1773). El "escribano" por ejemplo, introducía la pluma en el tintero y escribía un número limitado de palabras ("Sed bienvenidos a Neuchâtel"). Estos se conservan aún en el museo de arte e Historia de Neuchâtel, Suiza, y pueden observarse en la Ilustración 2.12.

Ilustración 2.12: "El dibujante", "La organillera" y "El escribano".

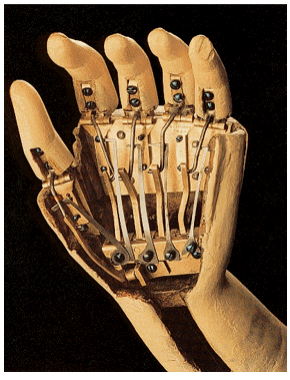


Ilustración 2.13: Mecanismo de una mano articulada.

Retomando de nuevo al inventor apasionado Wolfrang von Kempeler, debemos apuntar que fue el creador de uno de los más famosos autómatas de la historia, que a su vez, fue uno de los mayores fraudes de su tiempo por funcionar bajo el movimiento de un hombre, aún así impulsó la creación de autómatas jugadores de ajedrez hasta casi nuestros días. Hablamos de "*El Turco*" (Ilustración 2.14). Construido en 1769, estaba formado por una mesa donde estaba colocado un maniquí con forma humana vestido con vestiduras árabes. Una puerta en la parte frontal se abría y dejaba ver el supuesto mecanismo de funcionamiento del autómata. Este jugador fue una de las mayores atracciones de la época ya que, según contaban, era invencible. Viajó a lo largo de Europa y Estados Unidos aún después de la muerte de su creador.

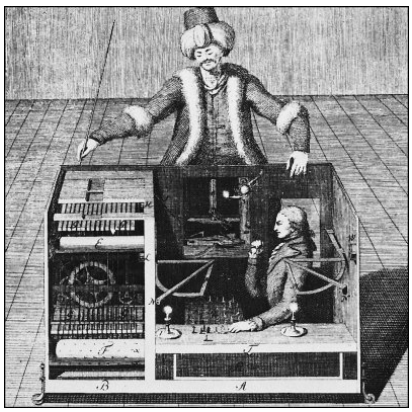


Ilustración 2.14: "El turco".

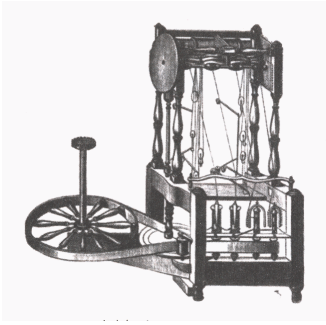
Sin embargo, todos estos no son más que juguetes pues no realizan un trabajo útil. Fue Blas Pascal quien con su "máquina calculadora" dio el primer paso hacia la verdadera automatización en el siglo XVII.

Ilustración 2.15: Máquina calculadora de Pascal.

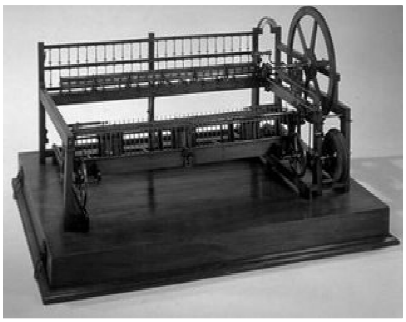
Otra familia muy importante en el desarrollo de los autómatas fueron los Maillardet (Henri, Jean-David, Julien-Auguste, Jacques-Rodolphe) que entre finales del siglo XVIII y principios del XIX, construyeron un escritor-dibujante con la forma de un chico arrodillado con un lápiz en su mano, el cual escribía en inglés y en francés y dibuja paisajes. Además, construyeron un mecanismo "mágico" que respondía a preguntas y un pájaro que cantaba en una caja.

Ilustración 2.16: "El dibujante".

A finales del siglo XVIII y principios del XIX se desarrollaron algunas ingeniosas invenciones mecánicas, utilizadas fundamentalmente en la industria textil, entre las que destacan;

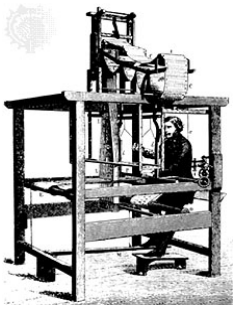


Hiladora giratoria de Hargreaves
(1770)



Hiladora mecánica de Crompton
(1779)

Telar mecánico de Cartwrigth
(1785)



Telar de Jacquard
(1801)

Ilustración 2.17: Ejemplos de maquinas textiles.

Jacquard basándose en los trabajos de Bouchon (1725), Falcon (1728) y del propio Vaucanson (1745), fue el primero en aplicar las tarjetas perforadas como soporte de un programa de trabajo, es decir, eligiendo un conjunto de tarjetas, se definía el tipo de tejido que se desea realizar. Estas máquinas constituyeron los primeros precedentes históricos de las máquinas de control numérico.

Algo más tarde que en la industria textil, se incorporan los automatismos en las industrias mineras y metalúrgicas. El primer automatismo que supuso un gran impacto social, lo realiza Potter a principios del siglo XVIII, automatizando el funcionamiento de una máquina de vapor del tipo Newcomen (Ilustración 2.18).

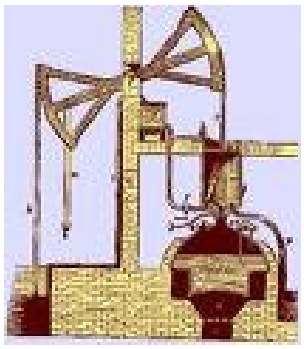


Ilustración 2.18: Mecanismo de Potter.

A partir de ahora, y a diferencia de los autómatas androides los automatismos dedicados a controlar nuevas máquinas industriales incorporan el concepto de realimentación. El ingeniero diseñador tenía una doble labor: realizar el proceso de diseño mecánico y desarrollar el automatismo, que en muchos casos era parte integrante de la mecánica de la máquina.

A partir de entonces es cuando el desarrollo de los automatismos es espectacular, en muchas máquinas se utilizan elementos mecánicos, como los programadores cíclicos (organillos) en los cuales se definía la secuencia de operaciones los cuales dieron lugar a los autómatas y robots actuales.

2.2 ROBÓTICA ACTUAL

De igual forma que se ha definido el término autómata para entender los antecedentes de la robótica, es necesario definir qué se entiende en la actualidad por ROBOT y los distintos tipos que existen.

Según la RAE [7] un robot es una máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas. Este término se dio a conocer a través de la obra teatral RUR (Rossum's Universal Robots) del dramaturgo Karel Čapek, proviene del Checo "*robota*", y significa trabajo, prestación personal o servidumbre, fue estrenada en 1921, y en ella se narra la historia de un científico que pretende liberar a la humanidad del trabajo físico a través de la invención de unos humanos artificiales que se encargan de realizarlo, sin embargo, al final de la obra estos robots se revelan y destruyen toda vida humana.

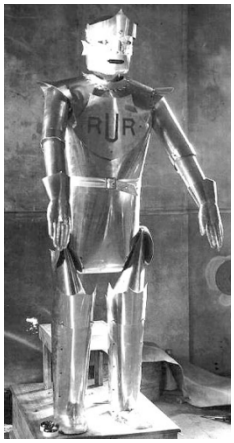


Ilustración 2.19: Robot de RUR de 1930.

Sin embargo, dejando a un lado la ficción, destaca Isaac Asimov(1920-1992) que acuñó la palabra "robótica", quien la enunció como la disciplina científica encargada de construir y programar robots, para él la robótica concentra 6 áreas de estudio: la mecánica, el control automático, la electrónica, la informática, la física y la matemática como ciencias básicas, y resumió la esencia de la misma en las "Tres leyes de la robótica" [8](Runaround, 1942) que dicen;

1. Un robot no debe dañar a un ser humano o, por su inacción, dejar que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto si estas órdenes entran en conflicto con la Primera Ley.
3. Un robot debe proteger su propia existencia, hasta donde esta protección no entre en conflicto con la Primera o la Segunda Ley.

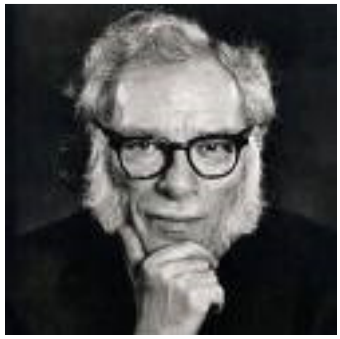


Ilustración 2.20: Isaac Asimov.

El pionero de la Robótica Industrial George Devol dio paso para mejorar las máquinas programables, y junto a Joseph Engelberger crearon la empresa UNIMATION. Éste creó en 1946 y patentó en 1948 un manipulador programable, flexible y de manejo sencillo que registraba señales eléctricas por medios magnéticos y podía reproducirlas para accionar una máquina mecánica. En ese mismo año C.Goertz del Argone National Laboratory desarrolló el primer telemanipulador del tipo maestro-esclavo que manipulaba elementos radioactivos reproduciendo los movimientos del operador sin riesgo para el mismo, el cual observaba a través de un cristal el resultado de las acciones, y sentía las fuerzas que el dispositivo esclavo ejercía sobre el entorno. Años más tarde Goertz aprovechó la tecnología electrónica y los servocontroles para sustituir la transmisión mecánica y desarrolló un telemanipulador con servocontrol bilateral.

G.C.Devol vuelve a la primera página cuando en 1954 desarrolla diseños para la transferencia de artículos programada, lo cual sienta la base para que en 1960 se implemente en un robot "Unimate" de transmisión hidráulica , utilizando los principios de control numérico para el control del manipulador. Por otro lado, en 1958 Ralph Moser diseñó "Handy-Man", fue uno de los pioneros en la telemanipulación, mientras, en 1959 se introdujo el primer robot comercial por Planet Corporation el cual estaba controlado por interruptores de fin de carrera.

A finales de los años 60 la visión artificial toma mayor relevancia y adquiere un nivel de desarrollo importante, este hecho queda demostrado cuando en 1968 el SRI (Standford Research Institute) crea un robot móvil llamado "Shakey", véase Ilustración 2.12, estaba previsto de gran diversidad de sensores así como una cámara de visión, sensores táctiles y podía desplazarse por el suelo.

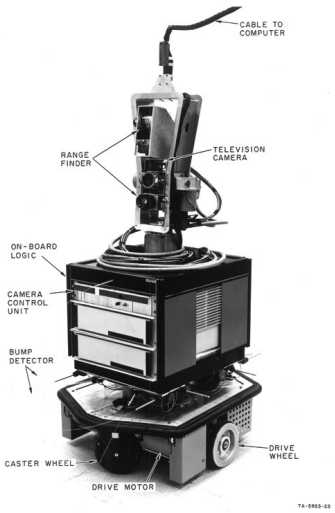


Ilustración 2.21: Robot "Shakey".

También fue en el SRI donde se desarrolló el primer lenguaje de programación de robots en 1973 denominado WAVE, a este le siguió el lenguaje AL en 1974; ambos dieron lugar al lenguaje VAL.

Gracias a estos lenguajes de programación la evolución de los robots industriales se vuelve espectacular, creando robots destinados a la realización de soldaduras, taladros, y sobre todo de montaje.

Actualmente los robots se rigen principalmente por control computacional, es decir, sus movimientos son dirigidos con gran precisión, permitiendo la repetición de acciones exactas. Dentro de las múltiples ventajas de este tipo de control se puede destacar la capacidad que adquiere el robot para aprender nuevas tareas, de tal forma que con la repetición de movimientos que no estaban programados se pueda crear una nueva tarea, el computador los recuerda y enseña al robot a realizarlos cuando deba.

Es en el sector industrial donde principalmente se destinan estos robots, pues una empresa debe estar bien automatizada si quiere ser competitiva en el mercado. Una de las industrias que más robots emplea es la del automóvil pues requieren complejas cadenas de montaje en las que los robots han sustituido la mano del hombre en tareas repetitivas y peligrosas como puedan ser la soldadura, la manipulación o la pintura, de esta forma, se consigue elevar el nivel productivo y minimizar los riesgos laborales de los empleados.

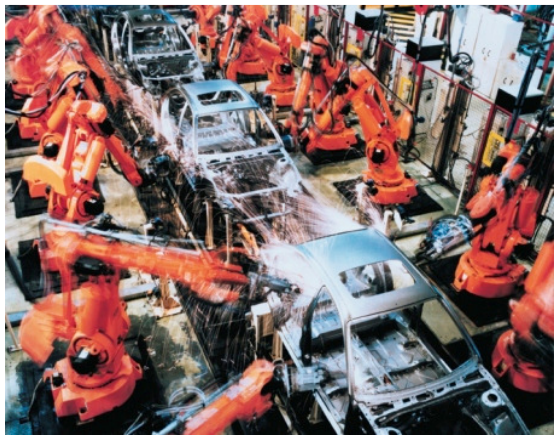


Ilustración 2.22: Cadena de montaje, sector automoción.

Aunque el campo industrial es el más amplio debido al gran volumen de robots que maneja, no resulta el único, existen multitud de actividades en la que los robots se han convertido en una pieza clave como pueden ser la limpieza de residuos tóxicos, la búsqueda de personas, la exploración espacial, marina, minera o la localización de explosivos.

En esta actividad destaca el trabajo realizado por los TEDAX (Técnicos Especialistas en Desactivación de Artefactos Explosivos) que se sirven de modelos teleoperados, es decir, guiados por control remoto para realizar su trabajo. Estos robots tienen diferentes mecanismos sensores que proporcionan información al operador, gracias a la cual es capaz de manipular objetos con gran precisión a una distancia que garantiza la seguridad del operador.

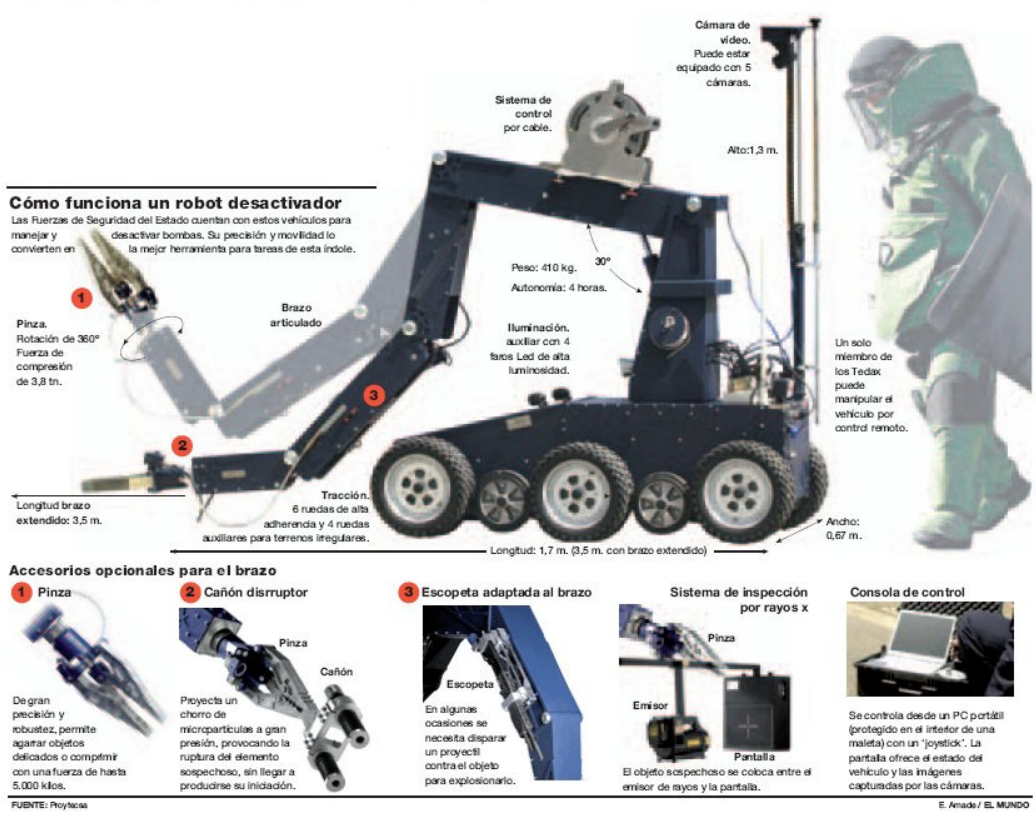


Ilustración 2.23: Robot artificiero de los TEDAX.

En la exploración marina podemos destacar a Nereus, fue el primer robot que exploró la Fosa de las Marinas en 1998, y en Julio de 2009, en este mismo lugar fue capaz de sumergirse a una profundidad imposible, llegó a los 10.902 metros [9]. No sólo se utiliza para la exploración de la vida y estructura marina, sino que también ha sido empleado en la búsqueda de náufragos como sucedió el verano de 2011 en Baja California [10].

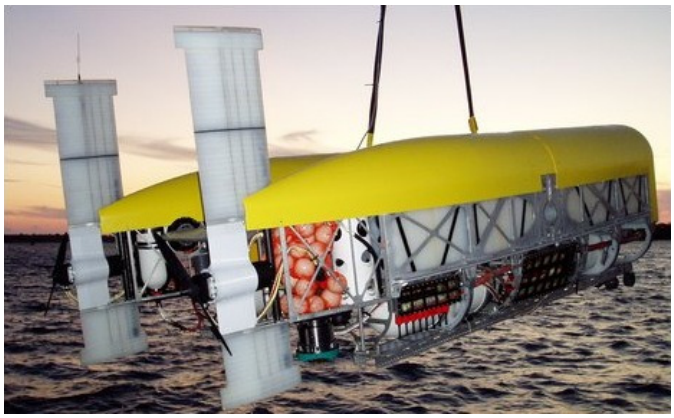


Ilustración 2.24. Nereus, robot subacuático.

La medicina es otro campo donde los robots tienen una parte esencial y son utilizados en procedimientos de cirugía, un ejemplo es el robot quirúrgico Da Vinci. Ilustración 2.25, de la compañía Intuitive Surgical, está dotado de cuatro brazos articulados, cada uno de ellos posee diferente instrumental y cámaras que permiten al cirujano realizar intervenciones quirúrgicas de gran precisión minimizando los riesgos para el paciente.

Ilustración 2.25. Robot quirúrgico Da Vinci de Intuitive Surgical.

En cuanto a los robots humanoides cabe destacar la gran evolución que se ha producido en los últimos tiempos, siendo uno de los más relevantes el robot ASIMO de la marca Honda, es autónomo y tiene diversos sistemas sensores destinados a la recogida de información del entorno con el que puede interaccionar en todo momento. Posee un sistema locomotor tan perfeccionado que es capaz de bailar, estrechar la mano de su interlocutor, subir y bajar escaleras, chutar balones y hasta transportar objetos, como bandejas, y dejarlos sobre una determinada superficie.

Ilustración 2.26: Evolución Robot ASIMO.

Actualmente el profesor Hiroshi Ishiguro de la Universidad de Osaka [11] es uno de los principales referentes en la robótica humanoide, entre sus creaciones, la más singular una réplica de él mismo, Geminoid, la cual ha usado para dar conferencias como si fuera él.

Ilustración 2.27: Distintas versiones de Geminoid.

Por otro lado y aprovechando las tecnologías desarrolladas para ASIMO, se han creado nuevos ingenios que hacen la vida más fácil a los seres humanos, como por ejemplo el exoesqueleto creado por esta misma marca para ayudar a caminar a personas de movilidad reducida, o el traje-robot diseñado por la Universidad de Agricultura y Tecnología de Tokio ideado para ayudar a levantar peso a los granjeros, reduciendo el esfuerzo sobre las articulaciones y la espalda en un 50%.

Ilustración 2.28: Exoesqueleto de Honda y Traje-Robot U.A.T.

2.3 CLASIFICACIÓN DE LOS ROBOTS

La utilidad y flexibilidad de un robot viene dado por la potencia del software empleado en el controlador. Dentro del ámbito de la robótica hay innumerables dispositivos diferentes entre sí, sin embargo podemos realizar una clasificación atendiendo a [12]:

2.3.1 Punto de vista histórico

- **Primera generación**

  A esta generación pertenecen los primeros autómatas, no tienen en cuenta el entorno y se caracterizan por la repetición de la tarea programada de manera secuencial.

  - Robots Play back; regeneran una secuencia de instrucciones grabadas. Estos robots tienen un control de Lazo Abierto. Como ejemplo un robot usado en la manipulación, servicio de maquinas o de soldadura por arco.

- **Segunda generación**

  Los robots pertenecientes a esta generación también son los llamados robots adaptativos. Estos robots actúan en consecuencia de la información recogida del entorno gracias a los sistemas de sensores de los que están dotados.

  - Robots controlados por sensores; tienen un control en Lazo Cerrado de movimientos manipulados y toman decisiones basadas en datos obtenidos por sensores. Un uso frecuente es la soldadura y pintura en cadenas industriales.

- **Tercera generación**

  Están programados en lenguaje natural, esta generación está formada por los robots con inteligencia artificial, están dotados de procesadores con una gran capacidad de cálculo, lo que les permite utilizar de forma muy eficiente la información obtenida a través de sus sensores para adaptarse al entorno y elaborar sus propios planes de acción. Hay que destacar que además poseen un sistema de aprendizaje que les permite superar situaciones imprevistas.

  - Robots controlados por visión; pueden manipular un objeto utilizando información proveniente de un sistema de visión. Usados en ensamblaje y desbarbado.

- **Cuarta generación**

  Son los robots empleados en los laboratorios con facilidad de adaptación tanto de sus acciones como de sus herramientas.

- Robots controlados adaptativamente; pueden adaptar sus acciones en función de los datos obtenidos por los sensores.

- **Quinta generación**

  Son el gran sueño pendiente, aún en desarrollo.

  - Robots con Inteligencia Artificial; usan técnicas de inteligencia artificial y son capaces de tomar decisiones propias para resolver problemas

A pesar de tantas generaciones existentes, hay que decir que los robots más extendidos en la actualidad son los representantes de la tercera y cuarta generación.

2.3.2 Nivel de control

- **Nivel de inteligencia artificial**; el programa acepta un comando de una acción y lo descompone en una secuencia de rutinas de bajo nivel según un modelo estratégico de tareas.

- **Nivel de modo de control**; los movimientos de sistema son modelados para lo que se incluye la coordinación dinámica entre los diferentes mecanismos, trayectorias planeadas y los puntos de asignación seleccionados.

- **Nivel de servosistema**; los actuadores controlan los parámetros de los mecanismos con el uso de una retroalimentación interna de los datos obtenidos por los sensores, la ruta es modificada sobre la base de datos que se obtiene de los sensores externos. Todas las detecciones de fallos y mecanismos de corrección son implementados en este nivel.

2.3.3 Arquitectura

- **Androides**: son robots humanoides, que pretenden imitar el comportamiento del hombre. Su utilidad en la actualidad es de solo experimentación, pero se busca que en un futuro sean capaces de interactuar con las personas y de ayudarlas, haciéndonos la vida más fácil y agradable. Algunos prototipos son tan cercamos como “Maggie” o “Rh-1”, desarrollados por el Robotics Lab de la Universidad Carlos III de Madrid, véase Ilustración 2.30 y 2.31. Otros más semejantes a los humanos son los ya mencionados *Geminoid.*

Ilustración 2.29: "Maggie".

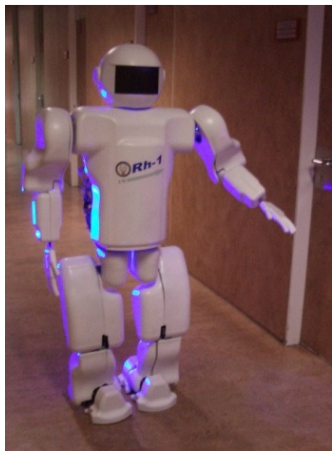


Ilustración 2.30: "Rh-1".

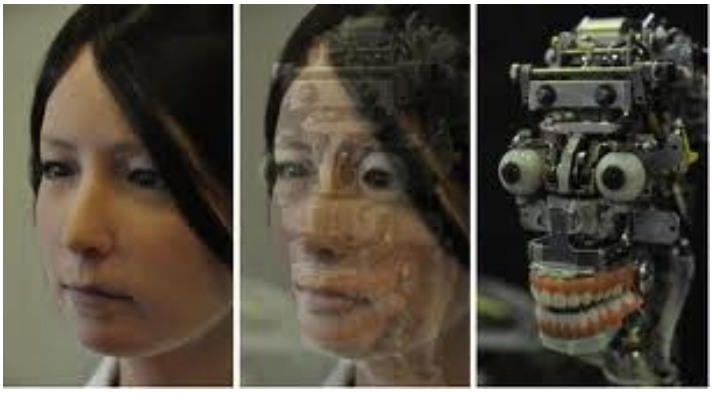


Ilustración 2.31: Geminoid.

- **Móviles**: se desplazan mediante una plataforma rodante clásica, que puede estar dotada de ruedas o de un sistema de orugas. Están muy extendidos en todo tipo de usos.

Ilustración 2.32: Robot móvil.

- **Zoomórficos**: emplean un sistema de locomoción diseñado imitando al de algunos seres vivos. Están en pleno desarrollo, pero principalmente se utilizan en misiones de exploración en otros planetas así como en el estudio de volcanes y entornos de difícil acceso.

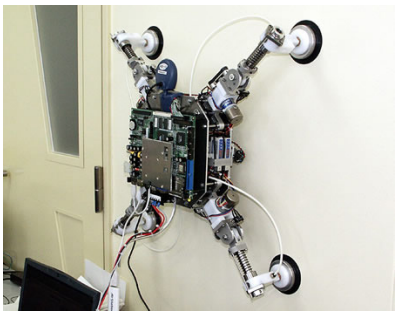


Ilustración 2.33: Robot zoomórfico.

- **Poliarticulados**: mueven sus extremidades con pocos grados de libertad. Se utiliza principalmente en la industria.

Ilustración 2.34: Robot, brazo articulado.

2.3.4 Nivel de lenguaje de programación

- **Sistemas guiados,** es el usuario quien dirige al robot a través de ciertos movimientos que debe realizar.

- **Sistemas de programación de nivel-robot,** el usuario escribe previamente un programa de computadora en el que especifica el movimiento y posteriormente analiza el recogido por los sensores.

- **Sistemas de programación de nivel-tarea,** en el cual el usuario especifica la operación por sus acciones sobre los objetos que el robot manipula.

2.3.5 Nivel de inteligencia

- **Dispositivos de manejo manual,** controlados por una persona.
- **Robots de secuencia arreglada,** se indica la secuencia que debe seguir.
- **Robots de secuencia variable,** donde un operador puede modificar la secuencia fácilmente.
- **Robots regeneradores,** donde el operador humano conduce el robot a través de la tarea.
- **Robots de control numérico,** donde el operador alimenta la programación del movimiento, hasta que se enseñe manualmente la tarea.
- **Robots inteligentes,** los cuales pueden entender e interactuar con cambios en el medio ambiente.

2.3.6 Federación Internacional de Robótica

- **Tipo A,** Manipulador que se controla manualmente o por control remoto.
- **Tipo B**, Manipulador automático pre-ajustado (PLCs, accionamiento neumático, eléctrico o hidráulico).
- **Tipo C**, Robot programable con trayectoria punto a punto.
- **Tipo D**, Robot capaz de adquirir los datos de su entorno y readaptar su función.

2.4 BRAZOS ARTICULADOS. ROBOTNIK

Un brazo articulado robótico es brazo mecánico programable, que forma un mecanismo por sí mismo, o forma parte de un robot de mayor complejidad. La funcionalidad principal es realizar los movimientos de un brazo humano. Está formado por articulaciones y eslabones, identificando las articulaciones como la interconexión de los eslabones, o partes del robot.

Los movimientos que pueden realizar estas articulaciones son rotacionales y translacionales o lineales tal como puede apreciarse en la Ilustración 2.35.

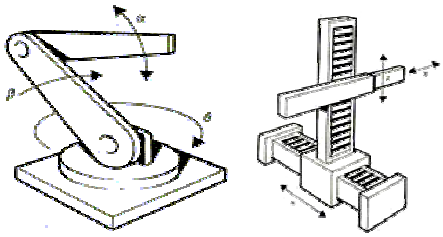


Ilustración 2.35: Movimiento rotacional y translacional.

En el mercado podemos encontrarnos brazos de los siguientes tipos[13]:

- Cartesiano: dispone de tres articulaciones prismáticas cuyos ejes coinciden con los ejes cartesianos.

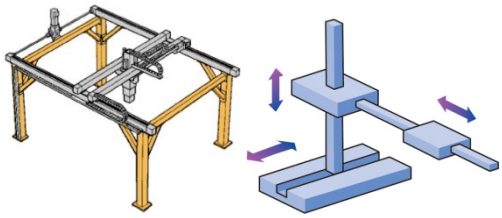


Ilustración 2.36: Robot cartesiano.

- Cilíndrico: sus ejes forman un sistema de ejes coordenadas cilíndricas.

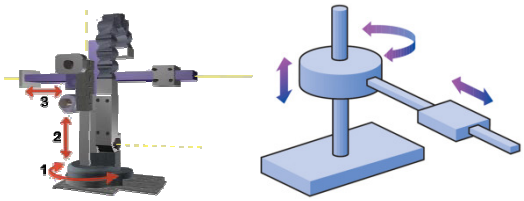


Ilustración 2.37: Robot cilíndrico.

- Esférico/polar: sus ejes forman un sistema de coordenadas polares.

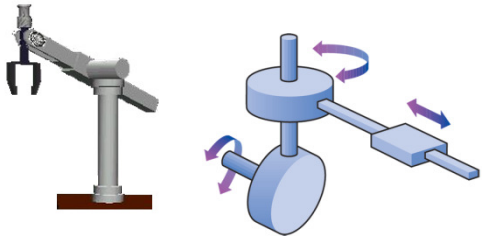


Ilustración 2.38: Robot esférico/polar.

- SCARA: tiene dos articulaciones rotatorias paralelas. Gira sobre la base y en el extremo tiene un eslabón deslizante en el eje Z. Se caracteriza por realizar movimientos en un plano.

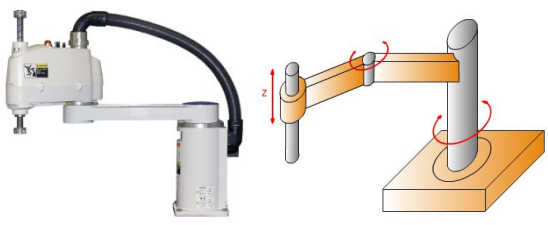


Ilustración 2.39: Robot SCARA.

- Articulado: mínimo tiene tres articulaciones rotatorias.

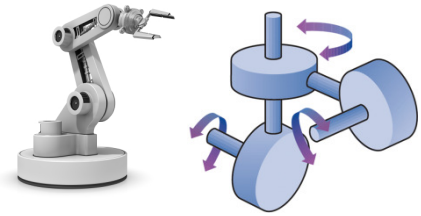


Ilustración 2.40: Robot articulado.

- Paralelo: sus brazos tienen articulaciones prismáticas o rotacionales concurrentes.

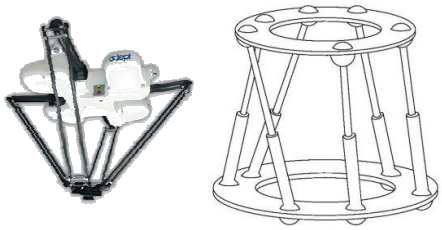


Ilustración 2.41: Robot paralelo.

- Antropomórfico: compuesto por dos eslabones rectos sobre una columna giratoria cuyas articulaciones son rotacionales, como su nombre indica, simula un brazo humano.

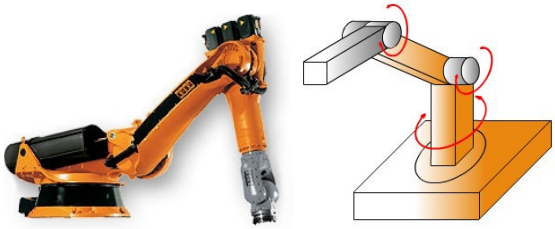


Ilustración 2.42: Robot antropomórfico.

ROBOTNIK: es el robot utilizado en este proyecto, es un "Brazo articulado" de tipo modular en configuración angular, sus movimientos son de revolución y tiene 6 GDL. Como se ha dicho, es modulable, lo que permite crear combinaciones de 1 a 7 GDL, conectarlo a cualquier tipo de sensor, compatible con PowerCube, Pinzas, PG070, PowerBall, SPB, y permite montar uno o más brazos en la misma plataforma. Robotnik, de ahora en adelante, tiene la posibilidad de incorporarle distintos accesorios como son la Mano Barrett, la Dextrous Hand o la Shadow Hand, aunque el comportamiento de estos elementos no es objeto de este estudio.

Ilustración 2.43: Mano Barrett, la Dextrous Hand y la Shadow Hand.

Los usos más frecuentes de Robotnik son la manipulación móvil de distintos elementos, su aplicación en la robótica humanoide, en la robótica médica o en la de servicio, a parte del carácter educativo que es el que concierne en este documento, de este último aspecto cabe destacar que el diseño de este robot está basado en un diseño original de la Universidad de Economía y Tecnología de Budapest (Departamento de Ciencias y Tecnologías de la Fabricación). Este brazo integra la etapa de potencia y la de control por lo que realmente está adecuado para su instalación en unidades móviles (plataformas móviles o androides, Ilustración 2.44).

Como grandes ventajas podemos encontrar;

- Alimentación: 24 VDC, por lo que no requiere inversor.
- Control mediante bus CAN: solo se necesita una tarjeta CAN instalada en un equipo PC, por lo tanto, no requiere de armario de control a diferencia de un brazo robot industrial.
- Configurable: las dimensiones de los enlaces se hacen de acuerdo con las necesidades del entorno y espacio de trabajo.

Por todo ello, resulta muy adecuado para instalar en las plataformas móviles.

Ilustración 2.44: Plataforma móvil con brazo Robotnik incorporado

Lleva integrados distintos elementos como son;

- Brazo robot integrado por servoaccionamientos PowerCube y los elementos de enlace entre los módulos.
- El elemento de ensamblaje con el chasis o la base donde se vaya a anclar.
- El elemento de ensamblaje con el efector final.
- Conexión y cableado.
- Tarjeta de comunicación CAM.

Los módulos, de la marca Shunk, trabajan como controladores distribuidos. El controlador maestro (PC de control) es el encargado de generar la secuencia del programa y el envío de referencias a cada uno de los ejes del sistema de articulaciones. El control de corriente, velocidad y de posición tiene lugar en cada uno de los módulos, así como las operaciones de supervisión de temperatura y control de parada. En cada uno de los ejes se encuentran instalados frenos electromagnéticos. También existe la posibilidad de cerrar los lazos de control (corriente, velocidad y posición) en el controlador externo.

Ilustración 2.45: Módulo Shunk.

En cada uno de estos módulos podemos encontrar un controlador y servoamplificador, motor sin escobillas (brushless), eje pasante por el que se introducen los cables, enconder absoluto para posicionamiento y control de velocidad, posibilidad de monitorización del rango, límites, temperatura y corriente, así como un freno magnético.

También dispone de distintos elementos o eslabones, los cuales, así como la teoría cinemática empleada en este estudio, se detallarán en el capítulo correspondiente.

CAPÍTULO 3

ENTORNO HARDWARE

En cuanto al entorno hardware, nos referiremos a dos aspectos, el hardware propiamente dicho, y los fundamentos en los que se basa la manipulación del mismo. En primer lugar debemos hablar de las características físicas generales, así como de los módulos hardware que componen el brazo robótico objeto de este proyecto, en segundo lugar, de las teorías cinemáticas en las que se basa su manipulación.

3.1 ASPECTOS GENERALES

Robotnik, como ya se ha indicado anteriormente, es un brazo robótico articulado con 6 grados de libertad, y su manipulación se entiende en este caso con objetivos formativos. Este brazo articulado presenta las siguientes características físicas generales:

- Alcance: de 400 a 1300 mm.
- Capacidad de carga útil en función del diseño (9Kg en la base del efector final con el alcance máximo).
- Peso reducido: aproximadamente 19Kg en una configuración de 6GDL.
- Velocidad máxima de 57 grados/s en la base y 360 grados/s en la muñeca.
- Motores servo de corriente alterna y frenos electromagnéticos.
- Repetitividad posicional de 0.5 mm.
- Electrónica de control mediante tarjeta CAN-PCI desde ordenador personal.

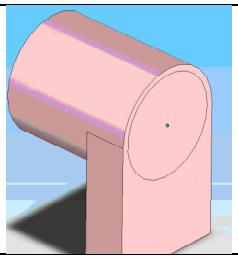
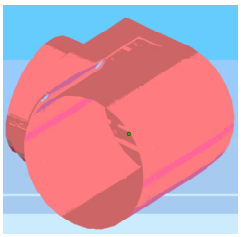
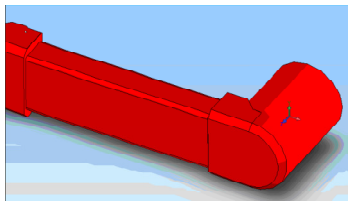
3.2 MODULOS Y ENLACES

Los módulos son de tipo rotatorio, son los ejes de las articulaciones y trabajan como controladores distribuidos, esto quiere decir que el ordenador actúa de maestro, crea y envía las referencias a los distintos módulos de esclavos. Para cada módulo se realiza independientemente el control de corriente, velocidad y posición, así como otras operaciones de supervisión y control de parada.

Entre las características físicas de estos módulos tenemos:

- Cada eje contiene su propio controlador y servo amplificador.
- Motores sin escobillas (brushless).
- Eje pasante que permite el paso interior de los cables (permite un diseño con protección IP).
- Encoder absoluto para posicionamiento y control de velocidad.
- Monitorización de rango, limites, temperatura y corriente.
- Requerimientos de potencia de 20A y 24VDC (en total 480W para todo el brazo).
- Integración inmediata con cualquier tipo de efector final: pinzas, manos robotizadas, taladros, útiles de soldadura, etc.
- Freno magnético integrado en todos los ejes.
- Arquitectura abierta (control de alto nivel sobre el movimiento de cada eje).

Los enlaces son las partes fijas que unen los distintos módulos, contamos con cinco elementos más la sujeción a la base, en la ilustración siguiente se pueden ver independientemente cada uno de estos:

ELEMENTO 1. UNIÓN 0-1	ELEMENTO 2. UNIÓN 1-2	ELEMENTO 2. UNIÓN 2-3
		
Este elemento está fijado a la base en la que se instala el brazo robótico y alberga el módulo 1.		

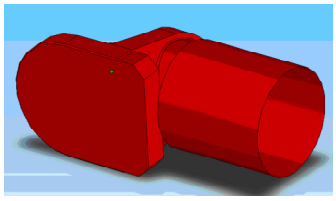
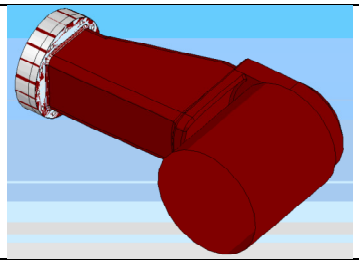
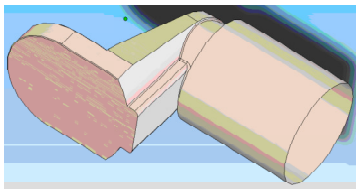
ELEMENTO 2. UNIÓN 3-4	ELEMENTO 2. UNIÓN 4-5	ELEMENTO 2. UNIÓN 5-6
		
		

Ilustración 3.1: Elementos del brazo Robontik.

3.3 ALCANCES Y LIMITACIONES

Este brazo modular de 6 g.d.l[1] presenta un amplio espacio de trabajo, sin embargo hay que tener en consideración que existen limitaciones angulares (Ilustración 3.2) para los distintos módulos con el fin de evitar algunas colisiones, evitar fallos técnicos referentes a los cables y minimizar las soluciones redundantes.

Joint	*Min*	*Max*
q1	*-180º*	*180º*
q2	*-126º*	*126º*
q3	*-100*	*100*
q4	*-180*	*180*
q5	*-90*	*90*
q6	*-180*	*180*

Ilustración 3.2: Límites angulares de las articulaciones.

[1] G.D.L: [23] Grados De Libertad, número de magnitudes que pueden variarse independientemente. Se necesitan tres grados de libertad para posicionar el efector terminal dentro de un entorno de trabajo tridimensional. Se necesitan otros tres para dirigir el efector terminal hacia cualquier dirección.

Con estas restricciones angulares tenemos el siguiente espacio de trabajo:

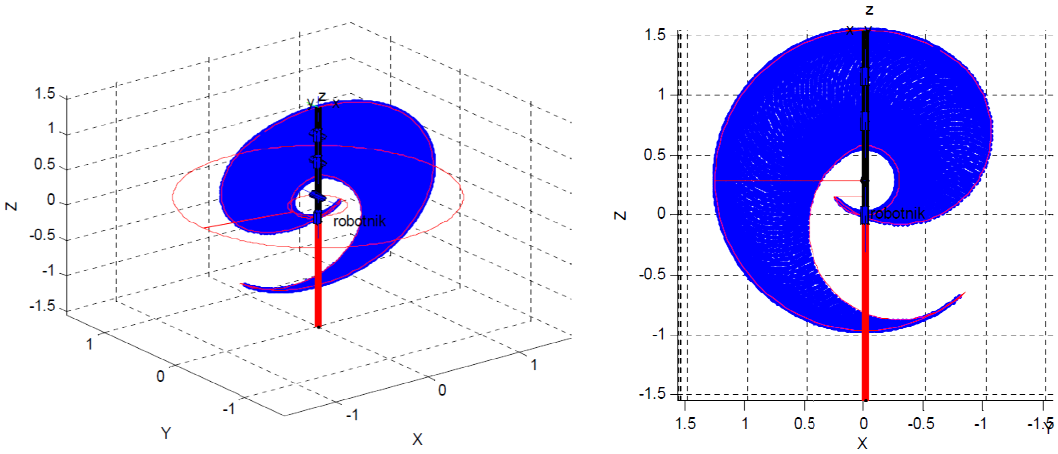


Ilustración 3.3: Espacio de trabajo de Robotnik.

El brazo modular implementa un algoritmo anticolisiones basado en la librería VCOLLIDE/RAPID[14], que es un algoritmo rápido para detectar colisiones bajo VRML[2]. Esta librería está diseñada para operar en ambientes con gran número de objetos geométricos formados por mallas de triángulos, realizando la detección de colisiones en dos etapas, una primera en la que se construyen los OBB's[3] para cada objeto con el fin de encontrar parejas de triángulos que intersecten, y una segunda en la que se comprueba que realmente estas parejas intersectan. Estas etapas están en otro componente, la librería RAPID. Existen tres diferencias principales entre la librería VCOLLIDE y la RAPID:

- VCOLLIDE conserva la información de dónde se encuentran los objetos en el entorno, así si no se mueven, sus localizaciones no se recalculan, al contrario que con RAPID.
- VCOLLIDE permite realiza una verificación simultánea de múltiples objetos, RAPID sólo de parejas.
- RAPID reporta la colisión entre triángulos mientras que VCOLLIDE reporta la colisión entre objetos.

[2] Virtual Reality Modeling Language. "Lenguaje para Modelado de Realidad Virtual"
[3] OBB's: hierarchical oriented bounding boxes

Por otro lado, hay que tener en cuenta las distintas configuraciones que se pueden adoptar a través de las cuales se puede llegar a la misma solución, así en la Ilustración 3.4 tenemos la diferencia entre configuración de "Hombro izquierdo" o la de "Hombro derecho":

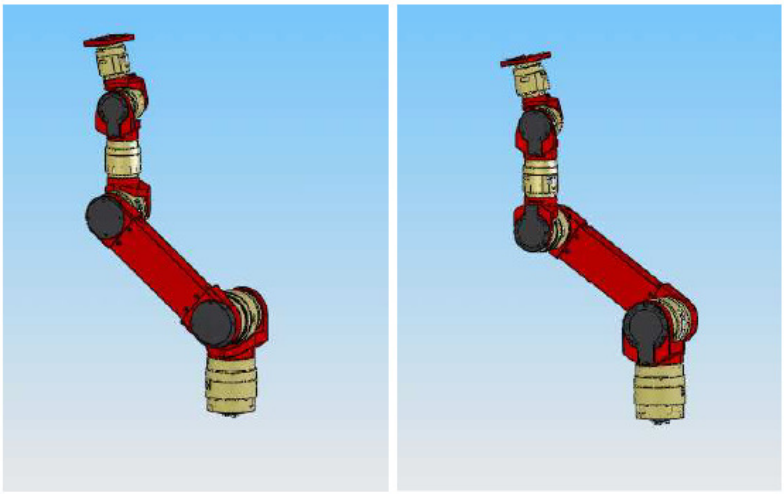


Ilustración 3.4: Hombro izquierdo & hombro derecho.

Si la configuración escogida es la de izquierdas, el q1 obtenido debe ser q1 + 180º. Seguidamente para el cálculo de q2 y q3 tenemos que el sen (θ3) puede ser positivo o negativo, dependiendo de la configuración del codo escogida, véase la Ilustración 3.5.

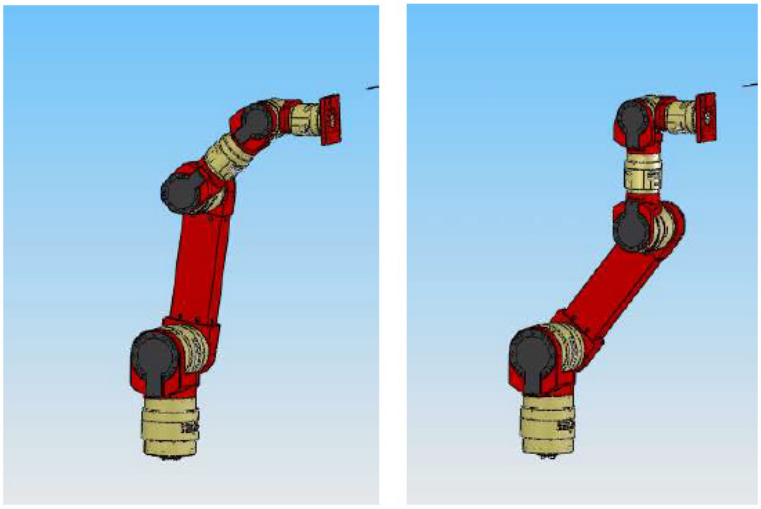


Ilustración 3.5: Codo abajo & codo arriba.

Por último, tenemos dos posibles soluciones dependiendo de la configuración del módulo 5, muñeca girada o no girada, según Ilustración 3.6.. En caso de que q_5 sea 0, tenemos infinitas soluciones, sin embargo se recomienda al menos mantener $5^o < q_5 < 10^o$ puesto que en un movimiento de trayectoria rectilínea pueden aparecer altas aceleraciones en los ejes 4 y 6.

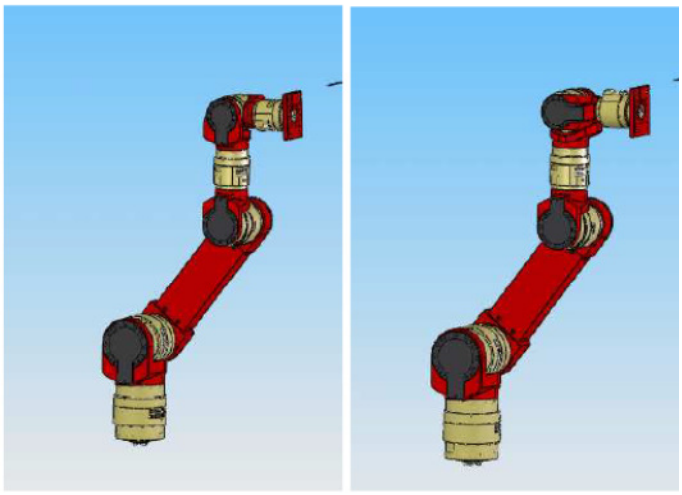


Ilustración 3.6: Muñeca girada & muñeca no girada.

En resumen, existen tres configuraciones distintas (hombro izquierda/derecha, codo arriba/abajo, muñeca girada/no girada), lo que nos permite tener $2^3 = 8$ combinaciones para llegar a la misma solución de posición y orientación del extremo del brazo articulado. Esto significa que para resolver un problema de cinemática inversa, es necesario conocer las tres configuraciones deseadas para seleccionar una de las 8 soluciones posibles.

3.4 CIRCUITO ELÉCTRICO Y FUENTE DE ALIMENTACIÓN

El circuito eléctrico está formado por la fuente de alimentación principal, la alimentación de los motores, y la tarjeta CAN, el esquema puede verse en la Ilustración 3.7..

La fuente de alimentación principal es de 24VDC 20A, esta alimentación puede suministrarte mediante dos fuentes distintas para evitar posibles ruidos, una para el control y otra para la alimentación del motor. Es muy importante conectar la masa de control con la masa del bus CAN. También es posible alimentar el brazo robótico con baterías, pero éstas deben cargarse estando desconectadas del robot para que el equipo electrónico no sufra daños. Y como cualquier equipo electrónico, si no se va a usar en un tiempo prolongado, se recomienda su desconexión.

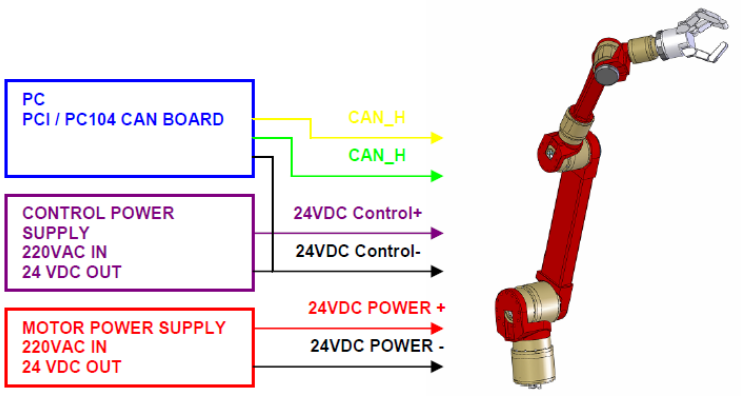


Ilustración 3.7: Circuito eléctrico.

3.5 SETA DE PARO DE EMERGENCIA

Como en cualquier dispositivo móvil, resulta imprescindible disponer de un interruptor de parada de emergencia. En este caso se trata de una seta independiente que el usuario debe tener siempre a mano para accionarla ante una situación de emergencia, si se diera el caso, se corta la alimentación a todos los módulos y se evitan daños en los actuadores y en el entorno.

También es recomendable accionarla para realizar cualquier trabajo de manipulación del robot en el que la total desconexión no sea necesaria, como puede ser el cambio del actuador extremo del robot o cambios en el entorno.

Para salir del estado de parada de emergencia es necesario seguir los siguientes pasos:

- Comprobar que la situación de peligro o motivo de parada se haya solucionado.
- Tirar del botón de parada de emergencia.
- Pulsar el botón de reinicio del panel de control.

Ilustración 3.8: Seta parada emergencia.

3.6 CINEMÁTICA DIRECTA. TEORÍA DE DENAVIT-HARTENBERG

A la hora de diseñar una interfaz gráfica se hace necesario conocer el comportamiento del dispositivo que se quiere manipular, de esta forma se puede determinar de una manera más eficaz qué tipo de funciones se quieren ofrecer a través de dicha aplicación.

En este caso, como ya se sabe, disponemos de un brazo articulado de 6 g.d.l., donde cada articulación es de tipo rotatorio con 1 g.d.l.. A la hora de planificar las características que debía tener la aplicación resultó importante poder indicar al usuario cuales eran las coordenadas cartesianas del extremo del brazo, de esta forma, era fácil reconocer dicha posición en tiempo real y ofrecer una información extra al usuario.

Para poder calcular estas coordenadas, ya que la librería no dispone de funciones específicas para tal caso, es imprescindible recurrir a la cinemática directa. La cinemática es la parte de la ciencia nos permite estudiar el movimiento de un robot sin tener en cuenta las fuerzas que sufre cada elemento del mismo (dinámica), sólo atendiendo a la posición, velocidad y aceleración. Si hablamos de cinemática directa, nos referimos a la técnica empleada para conocer la posición y orientación de cierto punto perteneciente a una estructura articulada a partir de las componentes fijas y transformaciones inducidas por las articulaciones de la estructura, siempre a partir de un sistema de coordenadas de referencia, siendo necesario conocer los valores de las articulaciones y los parámetros geométricos de los eslabones.

3.6.1 Denavit-Hartenberg

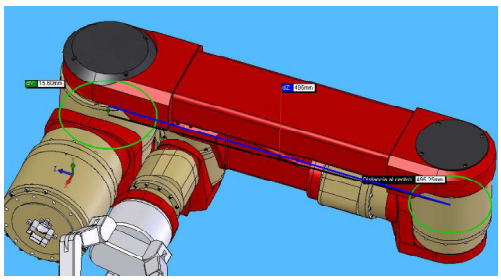
En 1955 Denavit y Hartenberg propusieron una forma de representación para describir la relación existente entre dos sistemas de referencia asociados a eslabones. Es un método matricial que liga a cada eslabón i, un sistema de referencia {Si} de manera sistemática. El paso de un sistema de referencia a otro se realiza mediante 4 transformaciones básicas que sólo dependen de las características geométricas del eslabón. Estas transformaciones están compuestas por la siguiente sucesión de rotaciones y traslaciones que relacionan el elemento i con el sistema de referencia {Si-1}:

1. Rotación alrededor del eje zi-1 un ángulo θi.
2. Traslación a lo largo del eje zi-1 una distancia di; vector di(0,0, di).
3. Traslación a lo largo del eje xi una distancia ai; vector ai(ai,0,0).
4. Rotación alrededor del eje xi un ángulo αi.

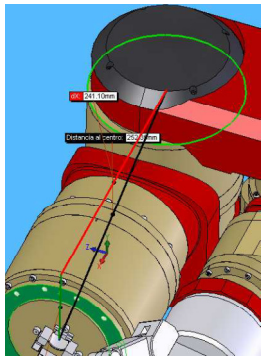
Es muy importante seguir este orden porque el producto matricial no es conmutativo, de esta forma nos queda la siguiente matriz de transformación donde di, αi, θi y ai son los parámetros D-H del eslabón i:

$$^{i-1}A_i = T = Rotz(\theta_i) \cdot T(0,0,d_i) \cdot T(a_i,0,0) \cdot Rotx(\alpha_i)$$

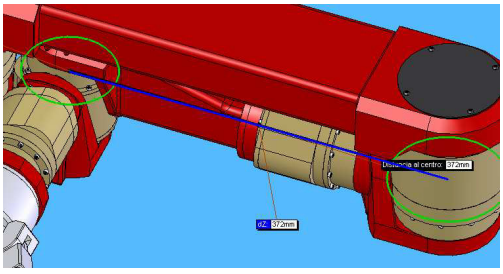
$$^{i-1}A_i = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i \cdot \sin\theta_i & \sin\alpha_i \cdot \sin\theta_i & a_i \cdot \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\sin\alpha_i \cdot \cos\theta_i & a_i \cdot \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



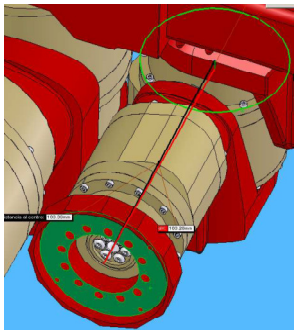
A2 = 0.495 mm



D1 = 0.2411 mm



D4 = 0.372 mm



D6 = 0.1832 mm

Ilustración 3.9: Parámetros Denavit – Hartenberg.

Para definir estas matrices de transformación es necesario seguir la siguiente metodología[15]:

1. **Numerar los eslabones:** se llamará "0" a la "tierra", o base fija donde se ancla el robot. "1" el primer eslabón móvil, etc.
2. **Numerar las articulaciones:** La "1" será el primer grado de libertad, y "n" el último.
3. **Localizar el eje de cada articulación:** Para pares de revolución, será el eje de giro. Para prismáticos será el eje a lo largo del cual se mueve el eslabón.
4. **Ejes Z**: Empezamos a colocar los sistemas XYZ. Situamos los Zi_{-1} en los ejes de las articulaciones i, con i=1,…,n. Es decir, Z_0 va sobre el eje de la 1ª articulación, Z_1 va sobre el eje del 2º grado de libertad, etc.
5. **Sistema de coordenadas 0:** Se sitúa el punto origen en cualquier punto a lo largo de Z_0. La orientación de X_0 e Y_0 puede ser arbitraria, siempre que se respete evidentemente que XYZ sea un sistema dextrógiro.
6. **Resto de sistemas:** Para el resto de sistemas i=1,…,N-1, colocar el punto origen en la intersección de Z_i con la normal común a Z_i y Z_{i+1}. En caso de cortarse los dos ejes Z, colocarlo en ese punto de corte. En caso de ser paralelos, colocarlo en algún punto de la articulación i+1.
7. **Ejes X:** Cada X_i va en la dirección de la normal común a Z_{i-1} y Z_i, en la dirección de Z_{i-1} hacia Z_i.
8. **Ejes Y:** Una vez situados los ejes Z y X, los Y tienen su direcciones determinadas por la restricción de formar un XYZ dextrógiro.
9. **Sistema del extremo del robot:** El n-ésimo sistema XYZ se coloca en el extremo del robot (herramienta), con su eje Z paralelo a Z_{n-1} y X e Y en cualquier dirección válida.
10. **Ángulos teta:** Cada θ_i es el ángulo desde X_{i-1} hasta X_i girando alrededor de Z_i.
11. **Distancias d:** Cada d_i es la distancia desde el sistema XYZ i-1 hasta la intersección de las normales común de Z_{i-1} hacia Z_i, a lo largo de Z_{i-1}.
12. **Distancias a:** Cada *ai* es la longitud de dicha normal común.
13. **Ángulos alfa:** Ángulo que hay que rotar *Zi*−1 para llegar a *Zi*, rotando alrededor de *Xi*.

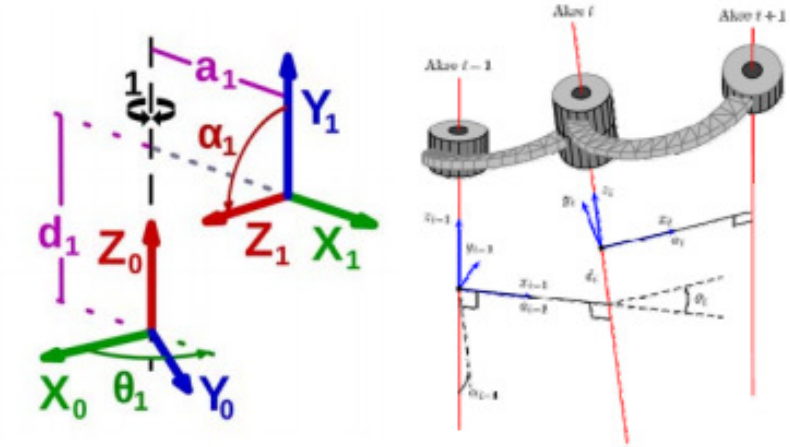


Ilustración 3.10: Ejemplo transición sistema S_{i-1} al S_i método Denavit-Hartenberg.

En este proyecto sólo se han considerado interesantes las coordenadas del extremo del brazo, siendo las únicas que se han calculado en las distintas pantallas de la aplicación. De esta forma, teniendo en cuenta los parámetros D-H de nuestro dispositivo, indicados en la Ilustración 3.9., y tras multiplicar las 6 matrices de transformación homogénea, podemos conocer las coordenadas cartesianas del extremo si tomamos los valores de la última fila de la matriz resultado:

X = D6*(sin(t5)*(sin(t1)*sin(t4) + cos(t4)*(cos(t1)*sin(t2)*sin(t3) - cos(t1)*cos(t2)*cos(t3))) - cos(t5)*(cos(t1)*cos(t2)*sin(t3) + cos(t1)*cos(t3)*sin(t2))) - D4*(cos(t1)*cos(t2)*sin(t3) + cos(t1)*cos(t3)*sin(t2)) + A2*cos(t1)*cos(t2)

Y = A2*cos(t2)*sin(t1) - D4*(cos(t2)*sin(t1)*sin(t3) + cos(t3)*sin(t1)*sin(t2)) - D6*(sin(t5)*(cos(t1)*sin(t4) - cos(t4)*(sin(t1)*sin(t2)*sin(t3) - cos(t2)*cos(t3)*sin(t1))) + cos(t5)*(cos(t2)*sin(t1)*sin(t3) + cos(t3)*sin(t1)*sin(t2)))

Z = D1 + D4*(cos(t2)*cos(t3) - sin(t2)*sin(t3)) + A2*sin(t2) + D6*(cos(t5)*(cos(t2)*cos(t3) - sin(t2)*sin(t3)) - cos(t4)*sin(t5)*(cos(t2)*sin(t3) + cos(t3)*sin(t2)))

CAPÍTULO 4

ENTORNO SOFTWARE

En primer lugar se hace necesaria la definición de software, y aunque existen multitud de ellas, se toma como la más adecuada la ofrecida por el estándar 729 del IEEE[4]:

Conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Dentro del entorno software empleado se diferencian dos sistemas distintos; el sistema operativo (SO) en el que se desarrolla todo el proyecto, y el software empleado para diseñar la interfaz gráfica de interacción con el robot. El primero se trata de MS Windows, el segundo del IDE[5] Visual Studio 2010(Visual C++) con lenguaje C++.

También es posible trabajar bajo LINUX en vez de MS Windows y desarrollar en otros lenguajes como Visual Basic o Visual C. En función de estos aspectos, las librerías empleadas se incluirán en el proyecto de forma particular para cada caso. En este proyecto se ha decidido trabajar con MS Windows por el conocimiento previo de este S.O, en cuanto a MS Visual Studio 2010, por ser una de las últimas versiones de los IDE de Microsoft, aportando gran funcionalidad y capacidad de trabajo.

4.1 MS WINDOWS

Microsoft Windows[16] engloba al conjunto de sistemas operativos que desarrolla y vende la empresa Microsoft. En 1985 introdujeron la versión 1.0 como un entorno operativo complementario (una extensión gráfica) al SO MS-DOS, y en 1990 se lanzó la primera versión

[4] Institute of Electrical and Electronics Engineers; Es la mayor asociación internacional sin ánimo de lucro formada por profesionales de las nuevas tecnologías

[5] IDE - Integrated Development Environment, programa informático compuesto por un conjunto de herramientas de programación, para uno o varios lenguajes de programación.

(3.0) basada en ventanas, debido a su interfaz gráfica de usuario[6] comenzó a utilizarse de forma generalizada abarcando más del 90% de la cuota de mercado.

En la siguiente tabla se puede ver un resumen de las distintas versiones que Microsoft ha vendido de este SO hasta el día de hoy.

Producto	Última versión	Lanzamiento	Comentarios
Windows 1.01	1.01	11/1985	Extensión gráfica de MS-DOS
Windows 2.03	2.03	11/1987	Por primera vez incluye ventanas que pueden solaparse
Windows 2.10	2.10	05/1988	
Windows 2.11	2.11	03/1989	
Windows 3.0	3.0	05/1990	Gran éxito, mejoras en la interfaz de usuario y multitarea
Windows 3.1	3.1	03/1992	Disponible para el público en general
Windows For Workgroups 3.1	3.1	10/1992	
Windows NT 3.1	NT 3.1	07/1993	SO Profesional
Windows For Workgroups 3.11	3.11	12/1993	
Windows 3.2	3.2	01/1994	
Windows NT 3.5	NT 3.5	09/1994	
Windows NT 3.51	NT 3.51.1057	05/1995	
Windows 95	4.03.1214(OS 2.5 ó 4.00.950C)	08/1995	Diseñado para sustituir a la versión 3.1, Workgroups y MS-DOS
Windows NT 4.0	NT 4.0.1381	07/1997	
Windows 98	4.10.1998	06/1998	En la primera versión tuvo fallos de lentitud y fiabilidad, pero se corrigieron en la 2ª edición (SE).
Windows 98 SE	4.10.2222	05/1999	
Windows 2000	NT 5.0.2195	02/2000	
Windows Me	4.90.3000	09/2000	
Windows XP	NT 5.1.2600	10/2001	La 1ª edición hizo énfasis en la funcionalidad de DVD, TV incluida grabación de TV y control remoto
Windows Server 2003	NT 5.2.3790	04/2003	Reemplaza la línea de productos de servidor de Windows 2000
Windows XP Professional x64 Edition	NT 5.2.3790	04/2005	
Windows Fundamentals for Legacy PC	NT 5.1.2600	07/2006	
Windows Vista	NT 6.0.6002	11/2006	Nuevas características. Especial atención a la seguridad.

[6] GUI - Graphical User Interface:

			Problemas de inestabilidad, sobredemanda de recursos de HW, alto costo y muy alta incompatibilidad con sus predecesores.
Windows Home Server	NT 5.2.4500	07/2007	
Windows Server 2008	NT 6.0.6002	02/2008	
Windows 7 y Windows Server 2008 R2	NT 6.1.7601	10/2009	Actualización incremental para ser compatible con aplicaciones y HW con los que W.Vista no era compatible
Windows Server 2012	NT 6.2	09/2012	
Windows 8	NT 6.2.9200	10/2012	Nueva interfaz gráfica

Tabla 4.1: Características familia Microsoft Windows.

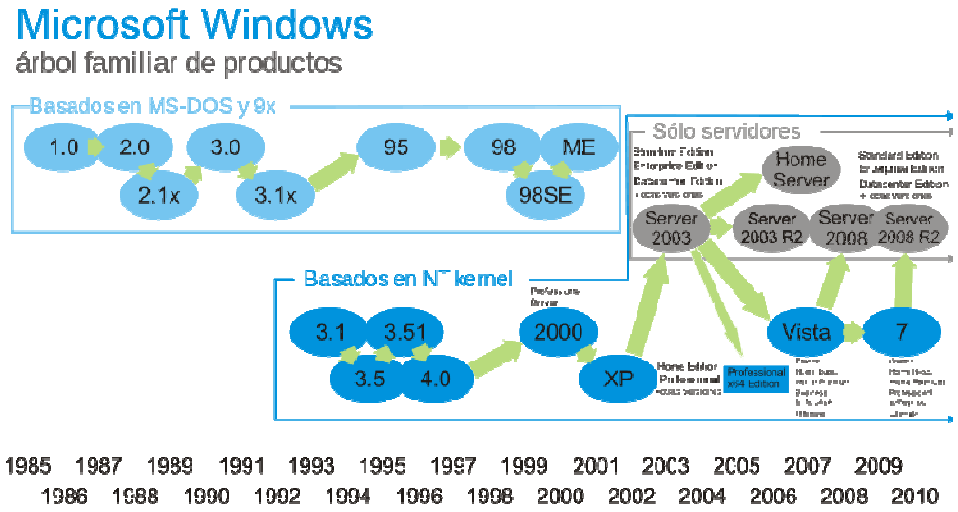


Ilustración 4.2: Familia Microsoft Windows.

En este proyecto se ha empleado el SO de Windows Vista, donde algunas de las nuevas aplicaciones que podemos encontrarnos son Windows Media Center que permite la grabación, visualización multimedia, Windows Aero, que es la nueva interfaz gráfica que ofrece la transparencia de las ventanas, permite una vista preliminar de las ventanas abiertas o cambiar entre ellas al presionar la tecla Windows y el tabulador. Incorpora también Internet Explorer 7 que permite la navegación por internet a través de pestañas rápidas e incluye mayor seguridad con advertencias antiphishing[7].

[7] Phishing: abuso informático que se comete mediante el uso de ingeniería social que pretende obtener información confidencial de forma fraudulenta.

4.2 LINUX

Se trata de un sistema operativo de 32 bits de libre distribución, desarrollado originalmente por Linus Torvalds[17]. Linux es distribuido bajo la Licencia General Pública (GPL) de GNU (GNU's Not Unix), lo cual significa que puede ser distribuido, copiado y modificado gratuitamente, a condición de no imponer ninguna restricción en sucesivas distribuciones. En pocas palabras: Linux es un sistema operativo gratuito.

GNU/Linux se refiere a la combinación del núcleo o kernel libre Linux (similar a Unix) con el sistema GNU. A las derivaciones de esta unión con la adición de aplicaciones específicas se denominan *Distribuciones*, cuyo objetivo es ofrecer ediciones que cumplan las necesidades de un grupo de usuarios. Algunas de las distribuciones más utilizadas entre los usuarios privados son Debian (7.0), openSUSE (11.4), Ubuntu (11.04) y Android (4.1).

Ilustración 4.3: Logos de Debian, openSUSE, Ubuntu y Android.

4.3 MS VISUAL STUDIO 2010

Microsoft Visual Studio 2010 es un IDE para el SO de Windows, que incorpora otros IDE como Visual C++, Visual C#, Visual J# y Visual Basic .NET. soportando sus correspondientes lenguajes de programación. Este ID es para la plataforma .NET, por lo que esta herramienta permite a los desarrolladores crear sitios, aplicaciones y servicios web, así como otro tipo de aplicaciones.

Esta versión es de las últimas del mercado y viene acompañada por .NET Framework 4.0.. Uno de los avances más importantes es la posibilidad de desarrollar aplicaciones para Windows 7 y para dispositivos con pantallas multitáctiles. Permite trabajar con múltiples monitores y acoplar las ventanas del programa en distintos puntos de la interfaz de trabajo.

El tipo de proyecto que hemos realizado es una aplicación de ventanas de formularios (Windows Forms Application) bajo Visual C++. Como puede apreciarse en la Ilustración 4.3, se pueden elegir entre otros tipos de lenguajes y otros tipos de aplicaciones.

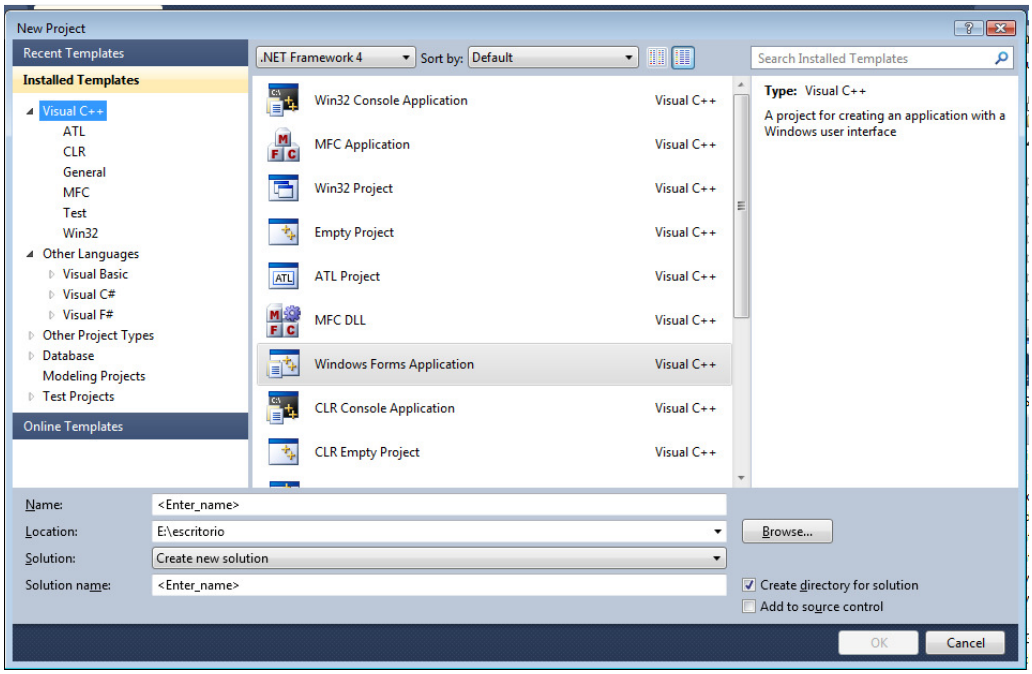


Ilustración 4.4: Configuración de un nuevo proyecto en MS VS10.

Dispone de una pantalla de inicio donde se puede acceder de forma rápida a los últimos proyectos, crear nuevos o acceder a otros más antiguos así como conectarse a Team Foundation Server[8] o acceder a las últimas noticias acerca del programa.

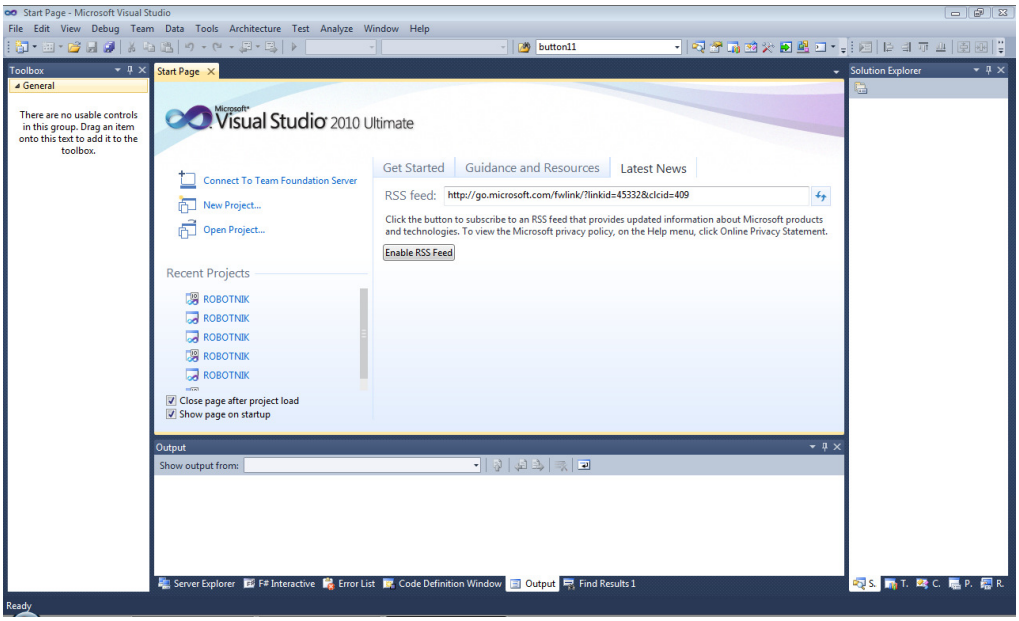


Ilustración 4.5: Página principal MS VS10.

[8] Team Foundation Server: Plataforma de colaboración y administración del ciclo de vida de las aplicaciones de Microsoft.

La aplicación dispone de un área de trabajo muy completa, en la que podemos destacar las siguientes ventanas:

- Ventana de trabajo (diseño & programación): hay dos tipos de ventanas, en una de ellas se realiza el diseño gráfico de los formularios, puede apreciarse en el título de la pestaña que se indica Form1.h [Desing]. El diseño se realiza de forma sencilla arrastrando los controles deseados de los disponibles en la lista de herramientas. Éstos pueden modificarse a través de código o de la ventana de propiedades. La otra ventana recoge el código de cada formulario, están definidas únicamente por el nombre del formulario, sin la anotación [Desing].

Ilustración 4.6: Ventana de trabajo. Diseño.

- Ventana de herramientas: en esta ventana se muestra un listado de los controles disponibles para incluir en los formularios, están clasificados por funcionalidad, o agrupados globalmente. Los controles que se han empleado son Button, RadioButton, Label, Panel, PictureBox, ContextMenuStrip y TabControl.

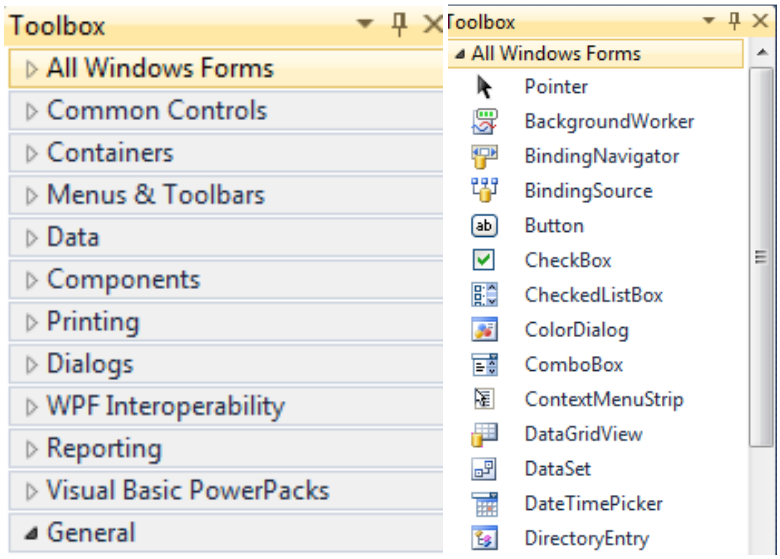


Ilustración 4.7: Ventana de herramientas.

- Ventana explorador: en esta ventana se tienen organizados todos los archivos y recursos que componen el proyecto. En la carpeta "External Dependencies" tenemos los archivos de los que depende el proyecto externamente, como las cabeceras stdio.h, time.h o math.h. En la carpeta "Header Files" se organizan los archivos .h; Form.h, m5apliw32.h, resource.h, stdafx.h. En la carpeta "Resource Files" se almacenan los recursos empleados, en este caso, sólo fotos. Y por último, en la carpeta "Source Files" tenemos los ficheros fuente .cpp.

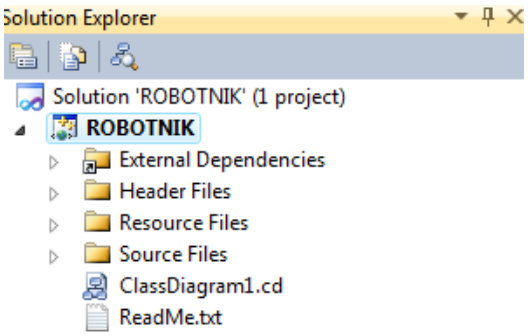


Ilustración 4.8: Ventana explorador.

- Ventana de propiedades (del control): en esta ventana se pueden modificar las propiedades características de cada control, algunas de éstas son comunes a todos los controles; Name, Size, Location o StartPosition, sin embargo otras características son exclusivas de cada control, en este caso, según la Ilustración 4.8 se trata del formulario Form1, y algunas de las propiedades que han sido modificadas son el icono personalizado que se podrá ver en la parte superior izquierda del formulario, se ha definido que se cargue el formulario en el centro de la pantalla sin que se pueda maximizar. También se ha cambiado la propiedad Text, que es el texto que aparecerá en la parte superior de la ventana, en este caso: ROBOTNIK.

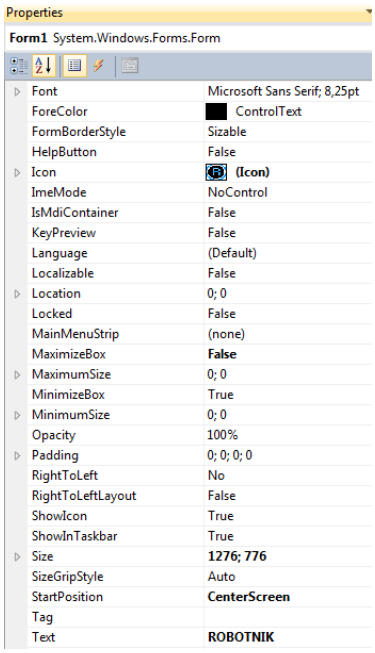


Ilustración 4.9: Ventana de propiedades.

- Ventana de salida: esta pantalla es imprescindible a la hora de ejecutar la aplicación, es aquí donde se muestran los errores y warnings a la hora de realizar el Debug de nuestro proyecto. De esta forma se pueden localizar fácilmente los problemas de código y solventarlos pues te indica el motivo de los mismos. En la Ilustración 4.9, puede verse por ejemplo los avisos de posible pérdida de información al convertir datos "double" a "float".

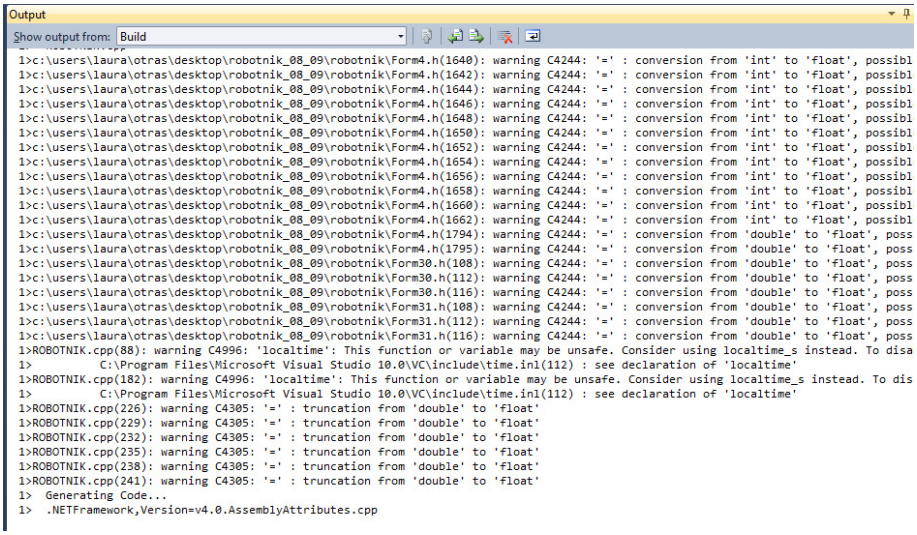


Ilustración 4.10: Ventana de salida.

4.4 LENGUAJES DE PROGRAMACIÓN. C++

Los lenguajes de programación son lenguajes formales que se emplean para expresar procesos que las computadoras pueden llevar a cabo. Un lenguaje se dice que es formal cuando sus símbolos primitivos y las reglas para unirlos están formalmente establecidos [18].

En base a los conocimientos previos y las posibilidades que ofrece Robotnik, el lenguaje de programación empleado en este proyecto es C++. Este lenguaje fue creado a mediados de los años 80 por Bjarne Stroustrup con el fin de extender el lenguaje de programación C con mecanismos que permitieran la manipulación de objetos, es por ello, que desde el punto de vista de la P.O.O[9] este es un lenguaje híbrido. También se entiende como un lenguaje multiparadigma que recoge los paradigmas de la programación estructurada, los de la orientada a objetos y de la genérica.

- Programación estructurada: está orientada a mejorar la calidad, claridad y tiempo de desarrollo de un programa para computadora utilizando solo subrutinas y tres estructuras; bucles (for, while), selección (if, switch) y secuencia. Además desaconseja el uso de la sentencia de trasferencia incondicional GOTO, que producía numerosos errores.
- Programación orientada a objetos: utiliza los objetos en sus interacciones para crear programas y aplicaciones informáticas. Incluye las técnicas de herencia, encapsulamiento, polimorfismo, cohesión y abstracción.

[9] P.O.O.: Programación Orientada a Objetos.

- Programación genérica: está centrada en los algoritmos en vez de en los datos. Busca crear funciones de forma generalista que se puedan usar ampliamente, esto se consigue parametrizando el desarrollo del programa y expresados o devueltos de la forma más simple posible. De esta forma, las funciones no sólo se pueden emplear en un programa de forma global, sino que mediante librerías se pueden utilizar en distintos programas.

En este proyecto, el lenguaje de programación se emplea con el fin de controlar física y lógicamente nuestro dispositivo, además de para realizar algoritmos con precisión. El control físico y lógico se realiza mediante el uso de variables y funciones que recogen el comportamiento del brazo robótico a través de la interfaz creada, las cuales permiten la interacción con el mismo. Las variables empleadas son de tipo global y local, las primeras tienen un ámbito general a todo el proyecto, se puede conocer su valor o modificarlo en cualquier punto del código, mientras que las segundas sólo son conocidas dentro de la función o formulario donde son declaradas.

4.5 LIBRERIAS, .ARCHIVOS .h y ARCHIVOS .cpp

Las librerías son un conjunto de implementaciones en un lenguaje de programación que definen perfectamente el comportamiento de una interfaz. Un programa se utiliza de forma autónoma, a través de su función principal, sin embargo una librería se diseña para ser utilizada por otros programas, lo que no quiere decir, que una librería dependa de otra para funcionar correctamente.

Hay dos tipos de bibliotecas, las estáticas y las dinámicas. Las primeras fueron las que se crearon en un inicio con el fin de actuar como un recipiente de ficheros de código objeto, que sólo se diferencian semánticamente de los ficheros objeto intermedios creados durante la compilación, durante este proceso estos ficheros son copiados y relocalizados, si fuera necesario, en el fichero final.

Sin embargo, las bibliotecas dinámicas son ficheros que contienen código objeto independiente de su ubicación, son requeridas y cargadas en tiempo de ejecución por cualquier programa que las invoque y no es necesario enlazarlas previamente.

4.5.1 Librería m5api.h

Para mover el brazo modular como una cadena cinemática, sus motores deben moverse independientemente, para ello se usa la librería m5api, suministrada por Schunk. Esta librería se puede compilar bajo distintos sistemas. En nuestro caso, hemos utilizado el sistema operativo Windows y el compilador Visual C++.

Operation sytem	Form	Supported Compilers
MS Windows 9x/NT/2000/XP	Dynamic Link Library (DLL)	Visual C/C++ Visual Basic National Instruments LabWindows CVI
SuSe Linux 6.4	Open Source	GNU C/C++
QNX 4.25	Open Source	Watcom C/C++ v. 10

En el Anexo I se pueden consultar las funciones con las que se puede controlar el funcionamiento del brazo robótico, entre ellas podemos destacar las siguientes:

- **Apertura y cierre de comunicaciones:**

Opening and closing the communication Interface	Function	Description
	PCube_openDevice	Opens the interface by specifiing an InitString. Result is a valid deviceID. See document "First steps" for further information on the InitString.
	PCube_closeDevice	Closes the interface by specifiing the deviceID.

PCube_openDevice: abre la interface especificando el String de inicio (ESD:0,1000), si la apertura es correcta almacena el Id del dispositivo en la variable dev.
PCube_closeDevice: cierra las comunicaciones con el dispositivo indicando el Id del mismo.

- **Administración:**

Administrative functions	Function	Description
	PCube_getModuleSerialNo	Retrieves the serial number of a module by specifiing deviceID and a moduleID.

PCube_getModuleSerialNo: almacena el número de serie del módulo especificado (1-6), en la variable correspondiente, serNoi, siendo i el identificador del módulo.

- **Posición:**

Retrieve position	Function	Description
	PCube_getPos	Retrieves the actual module position by specifiing deviceID and a moduleID. Result is the position in rad (rotary) or m (linear modules).

PCube_getPos: indicando el Id del dispositivo y el Id del módulo, almacena la posición en radianes en la variable correspondiente, posicioni, siendo i el identificador del módulo, si se trata de módulos giratorios, o metros para los módulos lineales.

- **Movimiento posicional:**

Ramp motion with specification of Target position	Function	Description
	PCube_movePos	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m. Prior to this call target speed and acceleration must be set using Funktionen PCube_setRampVel and PCube_setRampAcc resp. PCube_setRampVelInc and PCube_setRampAccInc.

PCube_movePos: indicando el Id del dispositivo y el Id del módulo deseado, así como la posición, se genera el movimiento tipo rampa para alcanzar dicha posición.

- **Parada de emergencia:**

Quick stop	Function	Description
	PCube_haltAll	Issues a Quick stop of all modules connected to the bus. For CAN-Bus only.

PCube_haltAll: indicando el Id del dispositivo se paran todos los módulos del mismo. Se emplea para la parada de emergencia en la Teleoperación-Por ángulos.

4.5.2 Librería math.h

Esta librería matemática es requerida para poder realizar las operaciones trigonométricas incluidas en las ecuaciones Denavit-Hartenberg para calcular las coordenadas "X Y Z" del extremo del robot.

Los datos deben introducirse en radianes, por eso a lo largo de la aplicación los ángulos siempre se recogen en tales unidades, así no se requieren conversiones y con ello se evita la posible pérdida de información.

4.5.3 Librería stdafx.h

Esta librería general recoge todas las referencias adicionales a las cabeceras del proyecto, es donde están definidas todas las variables y funciones globales del proyecto.

En cuanto a las variables externas tenemos las siguientes:

- Definición de constantes;
  extern double pi;
  extern const int NULO;

- Almacenamiento de la posición HOME
  extern float art1_home, art2_home, art3_home, art4_home, art5_home, art6_home;

- Almacenamiento de la posición INICIAL
  extern float Iniq0, Iniq1,Iniq2,Iniq3,Iniq4,Iniq5;

- Almacenamiento de la posición que se quiere alcanzar con la llamada de la función PCube_movePos(dev, i , qi);
  extern float q0, q1,q2,q3,q4,q5;

- Parámetros de Denavit Hartenverg
  extern double D1,A2,D4,D6;

- Almacenamiento de los valores por defecto de los ángulos límites de cada módulo.
  extern float q1min, q2min, q3min, q4min, q5min, q6min, q1max, q2max, q3max, q4max, q5max, q6max;

- Control de la teleoperación manual.
  extern bool articulaciones, controlp, controlv, sentido, pasop, pasov;
  extern bool a1, a2, a3,a4,a5,a6,cv,cp,pf,pm,pl,vf,vm,vl,izq,dch, pos, neg;
  extern float velocidadSeleccionada, pasoSeleccionado;
  extern unsigned long state1,state2,state3,state4,state5,state6;

- Control del menú de la aplicación, para relacionar Form padre - hijos.
  extern double a11,a12,a13,a14,a21,a22,a23,a24,a31,a32,a33,a34,a41,a42,a43,a44;

- Almacenamiento del valor del paso para el control manual por ángulo o por velocidad.
  extern double datapf, datapm, datapl, datavf, datavm, datavl;

- Almacenamiento del número de serie de cada módulo
  extern unsigned long serNo1,serNo2, serNo3,serNo4,serNo5,serNo6;

- Almacenamiento de la velocidad y posición de cada módulo
  extern float velocidad1, velocidad2, velocidad3, velocidad4, velocidad5, velocidad6;
  extern float posicion1, posicion2, posicion3, posicion4, posicion5, posicion6;

- Almacenamiento de la identificación del dispositivo (device) y el resultado de las funciones PCube_, las cuales serán correctas mientras que ret==0;
  extern "C" int ret,dev;

- Informe del chequeo que se guarda en un archivo .txt en la carpeta ArchivosTXT.
  extern int hora, minutos, segundos;
  extern int dia, mes, year;

- Detección durante el chequeo de errores de conexión.
  extern int errorModulos, errorDevice;

En cuanto a las funciones globales tenemos:

- Denavit – Hartenberg: utilizan el cálculo cinemático basado en la teoría de Denavit Hartenverg para el cálculo de las coordenadas cartesianas del extremo el último eslabón.
  double denavitHartenbergX (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);
  double denavitHartenbergY (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);
  double denavitHartenbergZ (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);

- Redondeo a "b" decimales los números tipo double "n" con los que se invoque la función.
  double fround(double n, int b);

- Realización del chequeo del robot y determinar si están todos los módulos conectados e inicializar sus variables. Se emplea al inicio de la aplicación y tras pulsar la seta de parada de emergencia pues hay que restaurar las comunicaciones.
  void RealizarChequeo(void);

- Función para refrescar el valor de las variables que almacenan la información de la posición de cada módulo.
  void CargarPosiciones(void);

- Función para asignar el valor de la posición actual de cada módulo a su correspondiente variable global "qi". Se utiliza en la Teleoperación Manual.
  void CargarQ(void);

4.5.4 Robotnik.cpp

Este archivo recoge la declaración de variables globales y la implementación de las funciones también de ámbito global. Es imprescindible que todas las variables externas que se han definido en el archivo stdafx.h también se definan en este, aunque sin la palabra clave "extern" delante.

Por otro lado, la implementación de las funciones globales debe hacerse en este punto, y tampoco puede faltar ninguna de las declaradas en el archivo stdafx.h.

Las funciones globales diseñadas en esta aplicación son las siguientes:

```
double denavitHartenbergX (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);
double denavitHartenbergY (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);
double denavitHartenbergZ (float Iniq0,float Iniq1,float Iniq2,float Iniq3,float Iniq4,float Iniq5);
double fround(double n, int b);
void RealizarChequeo(void);
void CargarPosiciones(void);
void CargarQ(void);
```

CAPÍTULO 5

COMUNICACIONES

En este tipo de proyectos es necesario hablar de las comunicaciones entre dispositivos, éstas se realizan mediante un protocolo por el que se establece un sistema de comunicación. Este protocolo de comunicación o protocolo de red define la forma en que la información en forma de mensajes circula por la red de dispositivos. Sin estas comunicaciones no sería posible interconectar el ordenador y el brazo robótico, por lo que las órdenes generadas en el primero no llegarían nunca al segundo, ni éste podría responder con el resultado de las mismas.

5.1 PROTOCOLO DE COMUNICACIONES

El funcionamiento del sistema se basa en un protocolo de comunicaciones, esto es un conjunto de reglas normalizadas para la representación, señalización, autenticación y detección de errores imprescindible para enviar información a través del canal de comunicación, en este caso el bus-CAN. El protocolo de comunicación tiene tres aspectos esenciales que hay que atender para el correcto funcionamiento del sistema, estas son;

- ✓ Sintaxis: se especifica cómo son y cómo se construyen.
- ✓ Semántica: qué significa cada comando o respuesta del protocolo respecto a sus parámetros/datos.
- ✓ Procedimientos de uso de esos mensajes: es lo que hay que programar realmente, y como tratar los errores.

El protocolo de datos se fundamenta en una serie de bits enlazados, los cuales sólo pueden adoptar valores "1" (transceptor[10] abierto) y "0" transceptor cerrado; conecta a masa. Nuestro bus-CAN utiliza un protocolo serie asíncrono del tipo CSMA/CD ("Carrier Sense Multiple Access with Collision Detection") basado en mensajes, donde la priorización y direccionamiento está en los propios datos transmitidos. Es un medio multiplexado y multicast, es decir, compartido, donde todos los aparatos pueden "hablar" de uno en uno si no detectan actividad (CSMA) y "escuchar", y así decidir si la información transmitida es aceptada o descartada. Si dos nodos quieren enviar información a la vez el método de arbitración CD, basado en prioridades resuelve la colisión, gracias a este arbitraje se evita la pérdida de tiempo

[10] *Transceptor: dispositivo que realiza funciones de envío y recepción de señales, no de forma simultánea, empleando elementos comunes del circuito para ambas funciones (semidúplex).*

y de información. Destacar por otro lado, que un nodo puede requerir información de otros nodos (RTR - Remote Transmit Request).

La arquitectura de protocolos CAN incluye tres capas: física, de enlace de datos y aplicación, además de una capa especial para gestión y control del nodo llamada capa de supervisor.

- Capa **física**: define los aspectos del medio físico para la transmisión de datos entre nodos, de éstos destacan los niveles de señal, representación, sincronización y tiempos en los que los bits se transfieren al bus. Los estándares ISO 11898 establecen las características que deben cumplir las especificaciones para la transferencia en alta y baja velocidad.

- Capa de **enlace de datos**: define las tareas independientes del método de acceso al medio y los tipos de tramas para el envío de mensajes. El intercambio de información se realiza en tiempo real, lo que requiere frecuencias altas de transmisión y retrasos mínimos.

- Capa de **aplicación**: Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación. Entre las capas de aplicación más utilizadas cabe mencionar CAL, CANopen, DeviceNet, SDS (Smart Distributed System), OSEK, CANKingdom.

- Capa de **supervisor**: este protocolo CAN dispone de un mecanismo autónomo de detección (aislamiento de fallos) y desconexión de un nodo defectuoso, el cual pudiera bloquear el funcionamiento del sistema completo, es por ello que cada nodo activo transmite una bandera de error cuando detecta algún tipo de error.

5.2 BUS DE COMUNICACIONES

La información tratada es transmitida a través del bus de comunicaciones, físicamente un bus es un conjunto de conductores que sirven para interconectar dos o más componentes funcionales de un sistema o varios sistemas, está formado por cables o pistas en un circuito impreso, dispositivos como resistencias y condensadores además de circuitos integrados. Eléctricamente, un bus es una colección de distintos niveles de tensión (corriente) específicos y señales que permiten a los dispositivos conectarse a él para trabajar correctamente.

Según la arquitectura de John von Neumann[19] existen tres tipos [20] de buses de comunicación:

- **Bus de datos**: este bus o línea permite el intercambio de información con la CPU. Cada instrucción de un programa y cada byte/bit viaja por este bus. El intercambio de información se realiza a través de líneas, que trasmiten los bits de forma aleatoria de manera que por lo general un bus tiene un ancho que es potencia de 2.

- **Bus de direcciones**: éste se basa en el intercambio entre la CPU (generalizando) y la Memoria Principal (MP). Este bus funciona sincronizado con el de datos y sirve

principalmente de guía para saber qué datos se envían/reciben a la CPU para su procesamiento. Hay que tener en cuenta que cuanto mayor sea el ancho de bus (bits) mayor rango de memoria direccionará, y, por lo tanto, con mayor cantidad de información podrá trabajar. Las líneas de direcciones por lo tanto son las encargadas de indicar la posición de memoria o el dispositivo con el que se desea establecer comunicación.

- **Bus de control:** son las encargadas de enviar señales de arbitraje entre los dispositivos. Entre las más importantes están las líneas de interrupción, DMA y los indicadores de estado.

Físicamente el bus CAN [21] está compuesto por cables, elementos de cierre, controladores y transceptores.

- **Cables:** se utilizan dos cables trenzados (bus diferencial) para obtener la inmunidad contra las EMIs (Interferencias Electromagnéticas). Unen todas las unidades de control que forman el sistema, y debido a la diferencia de tensión entre éstos, la información es transmitida. Un valor alto de tensión representa un 1 y un valor bajo de tensión representa un 0, siendo el mensaje a transmitir la combinación adecuada de unos y ceros.

  En uno de los cables, denominado "cable L" (Low) la tensión varía entre 0V y 2.25V, y en el otro, el cable H (High) lo hace entre 2.75V y 5V. El servicio sólo queda interrumpido cuando ambos cables se cortan, pues si se interrumpe la línea H o se deriva a masa, el sistema trabaja con la señal de Low con respecto a masa, si se interrumpa la línea L, ocurre lo contrario.

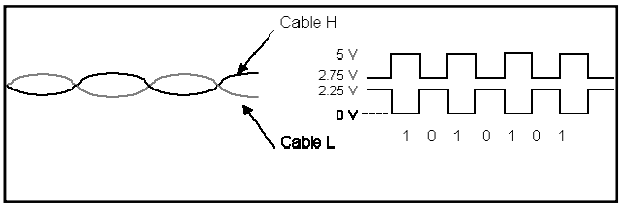


Ilustración 5.1: Cables.

- **Elementos de cierre:** Son resistencias conectadas a los extremos de los cables H y L del orden de los 120 Ω, pues aproximadamente, la impedancia característica de los cables tiene este valor. Su función es impedir el fenómeno de reflexión para evitar la perturbación del mensaje y que el bus se convierta en una antena. Sus valores se obtienen de forma empírica y permiten adecuar el funcionamiento del sistema a diferentes longitudes de cables y número de unidades de control abonadas. Se encuentran en el interior de algunas de las unidades de control del sistema por cuestiones de economía y seguridad de funcionamiento.

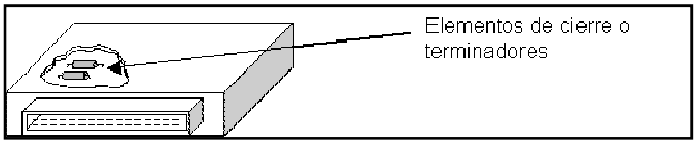


Ilustración 5.2: Elementos de cierre.

- **Controladores:** encargado de la comunicación entre el microprocesador de la unidad de control y el trasmisor-receptor, mediante el acondicionamiento de la información entrante y saliente entre ambos componentes. Trabaja a niveles muy bajos de tensión determinando la velocidad de transmisión de datos. Hay uno por cada unidad de control. Además, interviene en la necesaria sincronización entre las diferentes unidades de mando para la correcta emisión y recepción de los mensajes.

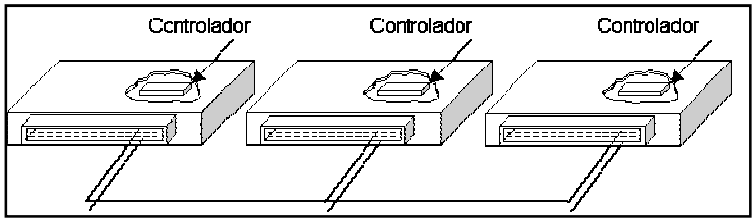


Ilustración 5.3: Controladores.

- **Transceptores:** es el encargado de la emisión y recepción de datos, además de acondicionar y preparar la información para que pueda ser utilizada por los controladores. Dicha preparación consiste en amplificar la señal cuando la información se vuelca en la línea y reduciéndola cuando es recogida de la misma y suministrada al controlador. Es un circuito integrado que está situado en cada una de las unidades de control abonadas al sistema, trabaja con intensidades próximas a 0.5 A y en ningún caso interviene modificando el contenido del mensaje. Funcionalmente está situado entre los cables que forman la línea Can-Bus y el controlador.

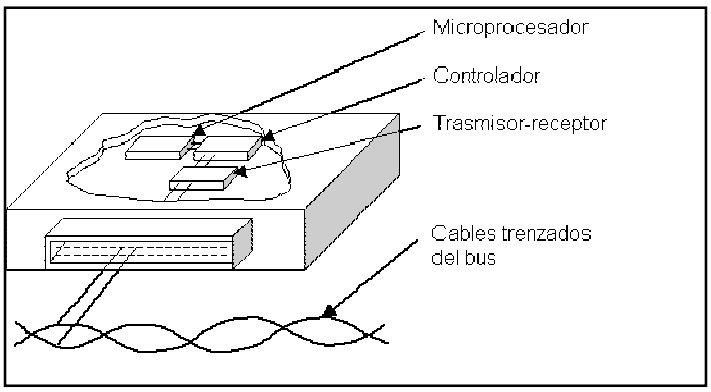


Ilustración 5.4: Transceptores.

5.3 BUS CAN

El bus CAN [22] tiene un único canal de comunicación entre las distintas conexiones, que teóricamente, no tienen límites, en la práctica, el número total de unidades conectadas al sistema está limitado por el tiempo y las cargas eléctricas. Los nodos se pueden añadir al sistema sin realizar cambios, y serán estos quienes interpreten si la información es para ellos o no, pues el identificador no indica el destino sino que describe el significado del mensaje, además, varios nodos pueden recibir la misma información a la vez (multicasting), no teniendo por qué utilizar toda la información procesada, sino que seleccionan la que deban utilizar. Este sistema cuenta con una gran robustez, pues garantiza que un mensaje enviado para distintos nodos en un mismo instante es aceptado por todos o alguno de ellos, siempre existe un control de la información enviada, recibida y rechazada. Por otro lado, igual que se envía información, también ésta se puede a distintos nodos; se envía una trama remota (RDR) de petición de traba y se contesta con ésta, ambas deben tener el mismo identificador. En el caso de coincidir el envío de una trama de datos y una remota, prevalece la primera. Para resolver casos de arbitraje, los transistores comparan el nivel del bits transmitido con el nivel de bus, si son iguales, la unidad puede enviar, si son distintos, la unidad pierde la prioridad y debe retirarse sin enviar otro bit.

Este bus de comunicaciones, aparte de ser robusto, es seguro, pues todos los nodos CAN implementan medidas especiales para la detección de errores, señalización y auto-chequeo. La detección del error consiste en la monitorización (comparación de niveles de bit), CRC (Comprobación de Redundancia Cíclica - es un tipo de función que recibe un flujo de datos de cualquier longitud como entrada y devuelve un valor de longitud fija como salida, es un código de detección de error), bit stuffing (inserción de bits no informativos en el mensaje) y chequeo de la trama de mensaje. En cuanto a la señalización de error y tiempo de restablecimiento, cabe destacar que los mensajes corruptos pueden ser reconocidos por cualquier nodo, éstos son anulados y reenviados de forma correcta, por otro lado, el tiempo de restablecimiento desde la detección del error es de 31bits. Además, los nodos CAN pueden distinguir también perturbaciones cortas y fallos permanentes, y si un nodo tiene problemas, avisa con un mensaje a las demás, si la situación es irreversible dicha unidad se desconecta, aunque sigue funcionando el resto del sistema.

5.3.1 Transmisión de datos

En el proceso de transmisión de datos se distinguen varias fases:

- **Suministro de datos:** una unidad de mando recibe información de los sensores que tiene asociados, su microprocesador pasa la información al controlador donde es gestionada y acondicionada para a su vez ser pasada al trasmisor-receptor donde se transforma en señales eléctricas.
- **Trasmisión de datos:** si el controlador encuentra libre el bus, transfiere los datos y su identificador junto con la petición de inicio de trasmisión, en caso de colisión con

otra unidad mandará quien tenga una prioridad mayor. A partir del momento en que esto ocurre, el resto de unidades de mando se convierten en receptoras.

- **Recepción del mensaje:** todas las unidades de mando reciben el mensaje y gracias al identificador determinan si usarán la información enviada y la procesarán, o no, entonces es ignorado.

5.3.2 Tramas

Como se ha comentado, la comunicación a través del bus CAN se realiza por mensajes, los cuales se pueden transmitir siempre y cuando el bus esté libre, y les caracteriza un identificador que aparte de definir el significado de la información, indica la prioridad que tiene ese mensaje. Los mensajes tienen una serie de campos de diferente tamaño (número de bits) y permiten llevar a cabo el proceso de comunicación, estos campos entre otras funciones, facilitan la identificación de la unidad de mando, indican el principio y el final del mensaje o muestran los datos. Dependiendo de la velocidad y de la unidad de mando que introduce los mensajes, el tiempo de vertido a la línea oscila entre los 7 y los 20 milisegundo.

Existen dos formatos de tramas de datos (data frame), las cuales difieren en la longitud del identificador; las tramas estándares (standard frame) poseen un identificador de 11 bits definidas en la especificación CAN 2.0A, y las tramas extendidas (extended frame) un identificador de 29 bits definidas en la especificación CAN 2.0B.

Para la transmisión y control de mensajes CAN, se definen cuatro tipos de tramas:

- ✓ Datos (data frame)
- ✓ Remota (remote frame)
- ✓ Error (error frame)
- ✓ Sobrecarga (overload frame)

Las tramas remotas también se establecen en ambos formatos, estándar y extendido, y tanto las tramas de datos como las remotas se separan de tramas precedentes mediante espacios entre tramas (interframe space). Una trama de datos siempre tiene una prioridad más alta que una trama remota. La trama remota se emplea para solicitar datos a otras unidades de mando o bien porque se necesitan o para realizar un chequeo.

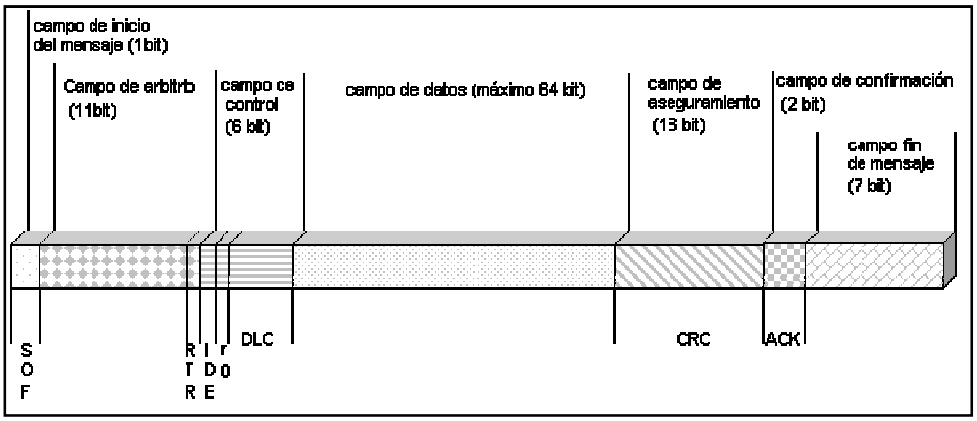


Ilustración 5.5: Trama.

Dentro del mensaje, encontramos los siguientes campos de información:

- **Campo de inicio del mensaje**: es un único bit, inicio del mensaje, del cual su flanco descendente es utilizado por las unidades de mando para sincronizarse entre sí.
- **Campo de arbitrio**: son 11 bits, constituyen el denominado Identificador, el cual, como ya se ha visto anteriormente permite reconocer a las unidades de mando la prioridad del mensaje. Cuanto más bajo sea el valor del identificador más alta es la prioridad determinando el orden en que van a ser introducidos los mensajes en la línea.
- Bit **RTR,** si su valor es 0 indica que el mensaje contiene datos, si es 1 implica que es una trama remota sin datos.
- **Campo de control**: informa sobre las características del campo de datos. Si el bit IDE es un 0, se trata de una trama estándar y cuando es un "1" es una trama extendida. Los cuatro bit que componen el campo DLC indican el número de bytes contenido en el campo de datos.
- **Campo de datos**: información del mensaje con los datos que la unidad de mando correspondiente introduce en la línea Can-Bus. Puede contener entre 0 y 8 bytes (de 0 a 64 bit).
- **Campo de aseguramiento (CRC** tiene una longitud de 16 bit y es utilizado para la detección de errores por los 15 primeros, mientras el último siempre es un bit recesivo (1) que delimita el campo CRC.
- **Campo de confirmación (ACK)**: está compuesto por dos bit que son siempre trasmitidos como recesivos (1). Todas las unidades de mando que reciben el mismo CRC modifican el primer bit del campo ACK por uno dominante (0), de forma que la unidad de mando que está todavía trasmitiendo reconoce que al menos alguna unidad de mando ha recibido un mensaje escrito correctamente. De no ser así, la unidad de mando trasmisora interpreta que su mensaje presenta un error.
- **Campo de final de mensaje (EOF)**: Este campo indica el final del mensaje con una cadena de 7 bits recesivos.
- **Bit stuffing:** estos bits no aportan información, simplemente se utilizan cuando en un mensaje se produce la singular situación de cinco bits seguidos iguales, de esta forma, tras cinco 0 insertará un 1, y viceversa. La unidad de mando que utiliza el mensaje, descarta un bit posterior a cinco bits iguales.

5.3.3 Ventajas

- Si ocurre un cortocircuito con masa, con positivo o mutuo entre los cables, el CAN-Bus pasa a la función de emergencia y cambia a funcionamiento monoalámbrico.
- Se necesitan menos cables para diagnósticos, porque todo el autodiagnóstico se gestiona a través de la unidad de control central.
- Permite disminuir notablemente el cableado, puesto que si una unidad de mando dispone de una información, esta puede ser utilizada por el resto de unidades de mando sin que sea necesario que cada una de ellas reciba la información individualmente.
- Las funciones pueden ser repartidas entre distintas unidades de mando, e incrementar las funciones de las mismas no presupone un coste adicional excesivo.

CAPÍTULO 6

EL PROBLEMA Y LA SOLUCIÓN. INTERFAZ GRÁFICA

Planteado el problema de diseño y creación de una interfaz gráfica que permitiera la manipulación de un brazo robótico modular con fines formativos, y tras haber analizado en los capítulos anteriores los distintos dispositivos y herramientas disponibles para desarrollarla, sólo queda exponer cual ha sido la solución alcanzada. Los objetivos principales que se habían marcado eran la teleoperación por ángulos, en la que se pudiera indicar unas posiciones concretas y el robot se desplazara hacia ellas, y la teleoperación manual, en la que por incrementos, el robot realizara un movimiento bien por incrementos de posición, o se desplazara a velocidad constante pero variable en función de los pasos definidos. Adicionalmente, se hacía necesaria una fase previa de control de comunicaciones y posibilidad de configurar los parámetros de trabajo, y como complemento, se añadió una parte de ayuda para el usuario con el fin de facilitarle información relevante acerca del propio robot y la programación empleada, por si en un futuro se decidiera ampliar esta aplicación con otras funcionalidades.

De esta forma, la interfaz gráfica que a continuación se describe es la solución final de este proyecto, es una aplicación con la que tras los análisis técnicos requeridos y las funcionalidades teóricas aplicadas, se puede manipular el brazo Robotnik. Es una interfaz de fácil manejo para el usuario, con las funcionalidades bien definidas y separadas en diferentes ventanas accesibles desde el menú, de tal forma que la persona que la utilice rápidamente pueda utilizarla con eficacia. Entre las tareas disponibles pueden distinguirse las siguientes:

- Acceso, creación de conexiones y chequeo del dispositivo.
- Configuración del dispositivo.
- Teleoperación, por ángulos y manual.
- Ayuda acerca de Robotnik, las librerías y programación empleadas.

6.1. INICIO Y CREACIÓN DE CONEXIONES

La aplicación se inicializa con la pantalla de acceso que se muestra en la Ilustración 6.1, a través del botón "ENTRAR". Al acceder a la aplicación lo primero que se requiere es realizar un chequeo para comprobar el estado del dispositivo, véase Ilustración 6.2, la correcta conexión de los módulos y su posición, a parte de otra información como el tipo de módulo, número de serie o los límites angulares.

Este chequeo cubre dos funciones, abrir las comunicaciones con el dispositivo a través del bus CAN y chequear el estado de los módulos, y por otro, generar un informe y guardarlo en el archivo Chequeo.txt ubicado en una carpeta de la propia aplicación para poder consultarlo o imprimirlo. Un ejemplo del mismo se puede consultar en el Anexo III.

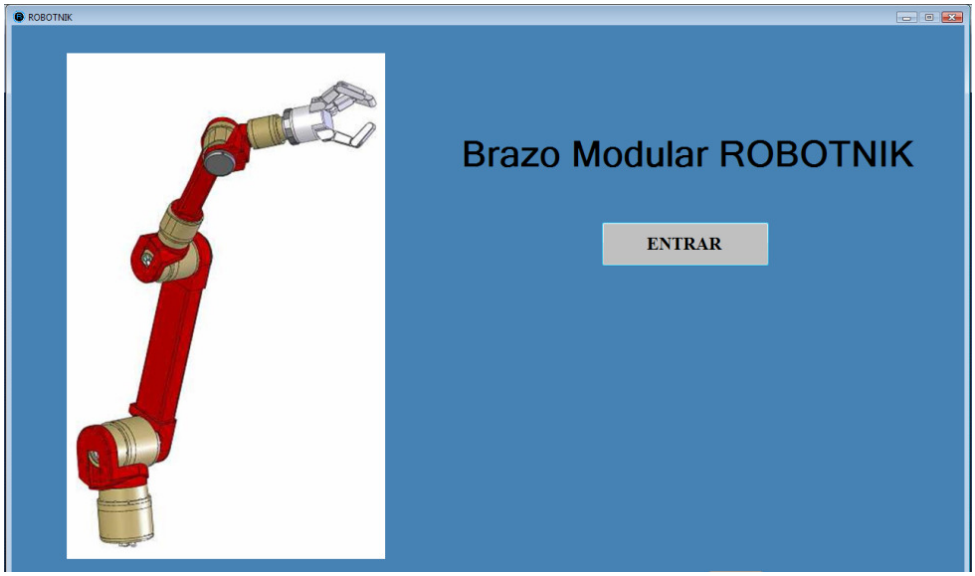


Ilustración 6.1: Ventana de acceso a la aplicación.

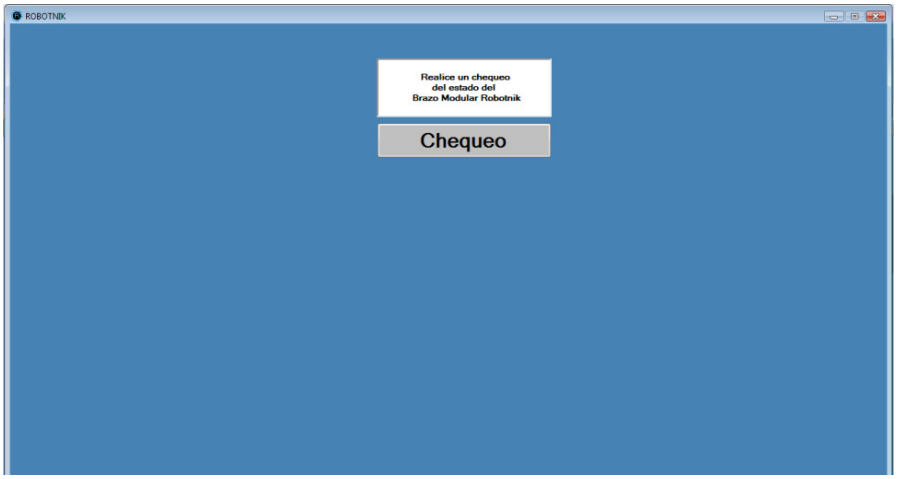


Ilustración 6.2.: Ventana solicitud de chequeo inicial.

En primer lugar se abren las comunicaciones con el dispositivo, se prepara la cabecera del informe con la fecha y hora de inicio del chequeo, y si las comunicaciones han sido correctas, se verifica el estado de cada módulo, sino, se graba un mensaje de error, se indica la hora de finalización del chequeo y se guarda el archivo Chequeo.txt. Si la conexión con el dispositivo se ha producido sin errores, antes de finalizar el informe del chequeo en el que se refleja la información del mismo, se verifica cada módulo individualmente, por cada uno, si se establece conexión se registra la posición del mismo y el número de serie, sino se muestra un mensaje de error.

Para poder acceder a las funcionalidades de la aplicación es necesario que tanto el dispositivo como cada módulo estén bien conectados y comunicados, de lo contrario sólo se puede repetir el chequeo o salir de la aplicación, véase Ilustración 6.3. De haber sido correcta esta fase previa (Ilustración 6.4), se habrán inicializado las distintas variables del programa, como los valores predeterminados de la posición Home, la posición Inicial para la teleoperación por ángulos o los de velocidad y posición para la teleoperación manual.

Ilustración 6.3: Ventana resultado del chequeo con errores en módulos 1 y 4.

Ilustración 6.4: Ventana resultado del chequeo sin errores.

6.2. CONFIGURACIÓN

En esta pantalla se pueden modificar los valores de las distintas variables de trabajo como puede verse en la Ilustración 6.5. Es posible cambiar los ángulos de la posición Home e Inicial, todos o alguno, éstos deben introducirse en radianes, con el signo indicado en la primera posición, y como máximo tendrá un decimal de precisión.

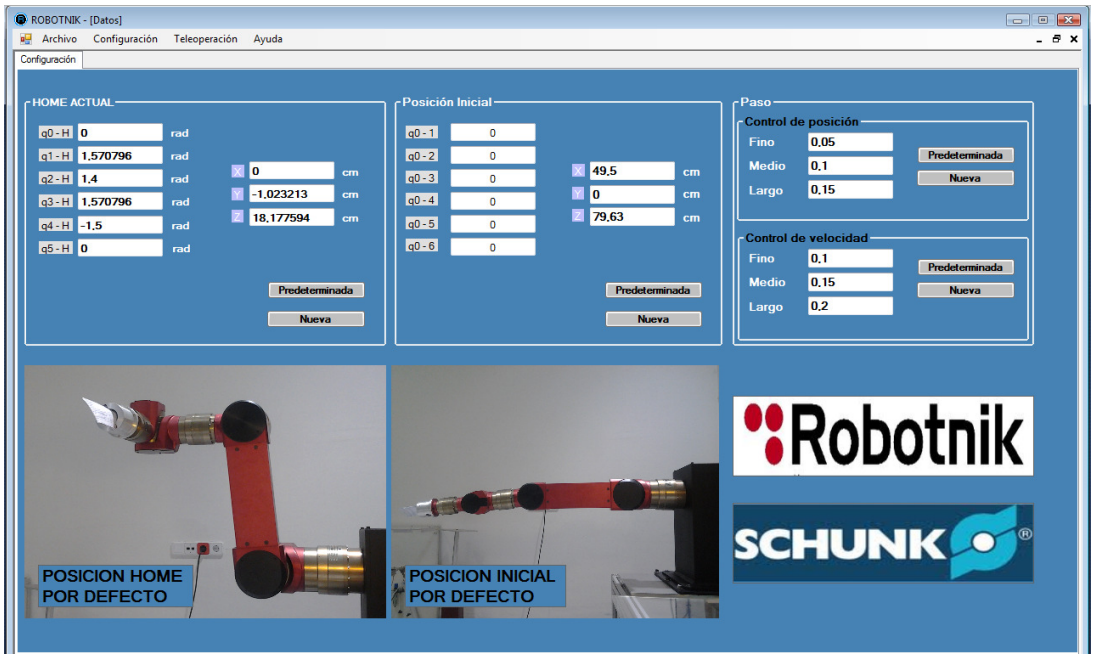


Ilustración 6.5: Ventana de configuración.

Para tomar esta decisión se han tenido en cuenta las siguientes relaciones;

1 rad = 28,647 grados

0,1 Rad = 2,8647 grados

0,01 rad = 0,2867 grados

Dado que tener en cuenta dos decimales apenas supone variación de posición, para las aplicaciones formativas a las que se destina este proyecto, era suficiente con indicar un solo decimal, puesto que dos decimales en la mayoría de los casos no resulta práctico.

Inicialmente se cargan los datos de la configuración predeterminada, si se desea seleccionar una nueva, hay que pulsar el botón correspondiente para que se abra la ventana auxiliar donde se podrán introducir todos o alguno de los datos que se quieren cambiar. En esta ventana se indica el formato en que deben ser introducidos, teniendo en cuenta que los valores deben pertenecer al rango de trabajo que posea cada articulación. Si se realiza correctamente se podrá observar que en el apartado correspondiente se muestras éstos nuevos datos, si no, se mostrará un mensaje de error indicando que debe revisar la notación (Ilustración 6.7). Este procedimiento es idéntico para el cambio de las posiciones Home e Inicial y puede observarse en la Ilustración 6.6.

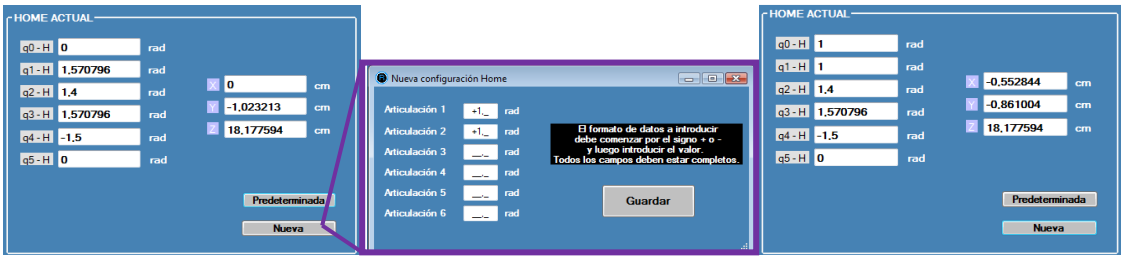


Ilustración 6.6: Secuencia de cambio de datos de la posición Home.

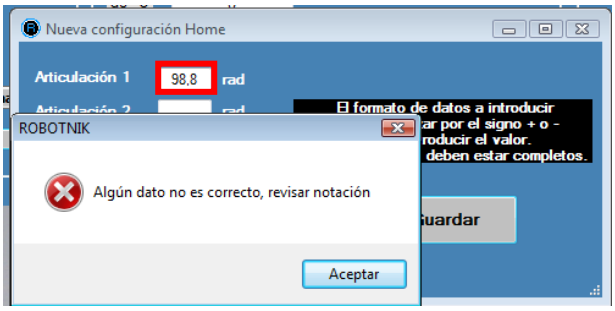


Ilustración 6.7: Mensaje de error posición 1fuera de rango.

En cuanto a la configuración de los datos de paso o incrementos, destacar que el funcionamiento es el mismo tanto para el paso de posición como para el de velocidad. Como puede seguirse en la Ilustración 6.8, se pulsa el botón de "Nueva" con el que se abrirá una ventana auxiliar donde se podrán poner los pasos deseados, se pueden modificar todos o sólo alguno de los tres.

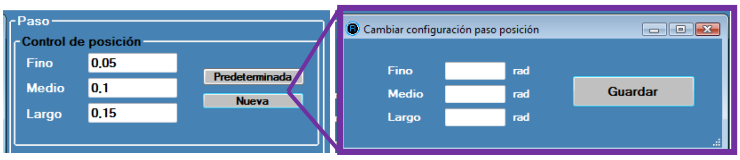


Ilustración 6.8: Cambio configuración paso de posición.

En cualquier caso, para cada una de los cuatro apartados de configuración, siempre se puede volver a cargar la configuración predeterminada pulsando el botón correspondiente.

6.3. TELEOPERACIÓN POR ÁNGULOS

El objetivo de esta pantalla (Ilustración 6.9) es teleoperar el brazo robótico indicándole las posiciones a las que debe llegar cada módulo. Se le pueden indicar todas las posiciones a la vez o sólo las deseadas, varias o de una en una.

Para introducir los datos de las posiciones, es necesario seleccionar mediante el Radio Button correspondiente el módulo que se quiere tratar (Ilustración 10-15), sólo se podrá modificar ese valor, y en el Picture Box de la izquierda se cargará una imagen con dicha articulación remarcada en la que se indica el sentido de giro para darle mejor soporte al usuario.

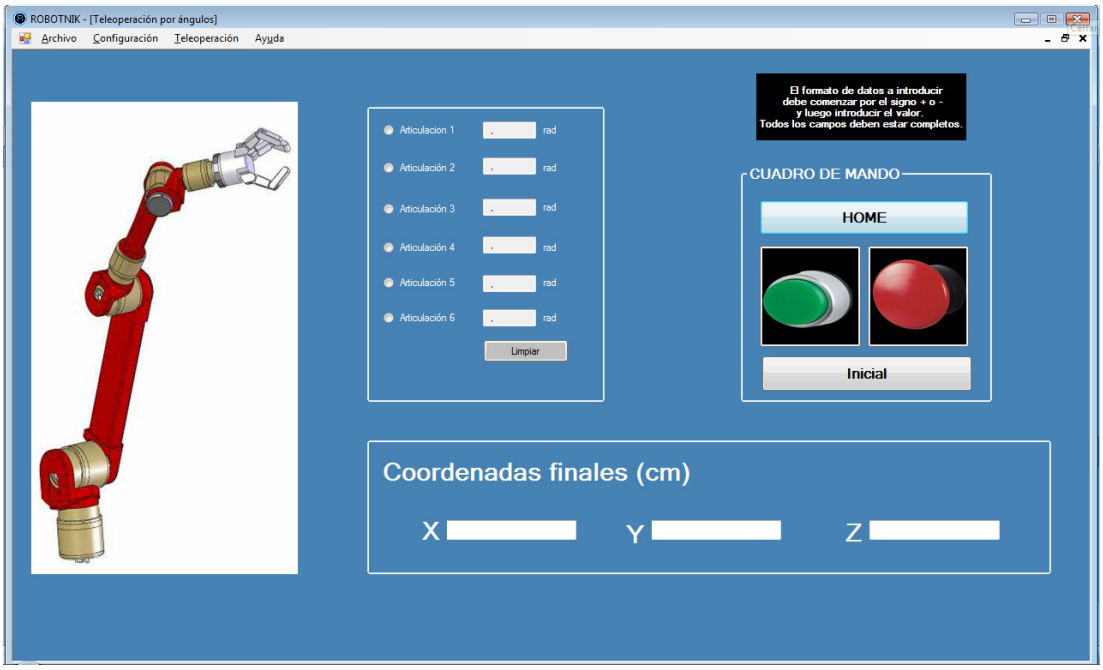


Ilustración 6.9: Ventana resultado del chequeo sin errores.

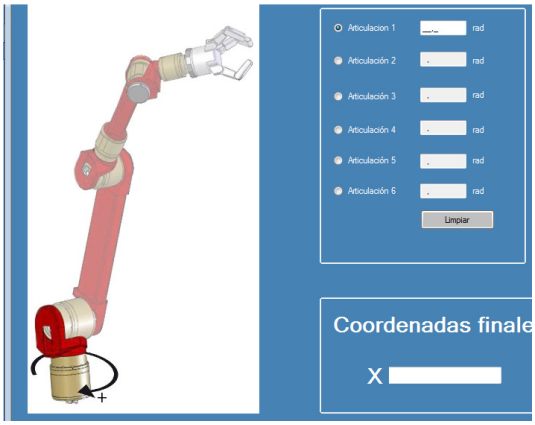


Ilustración 6.10: Módulo 1 seleccionado.

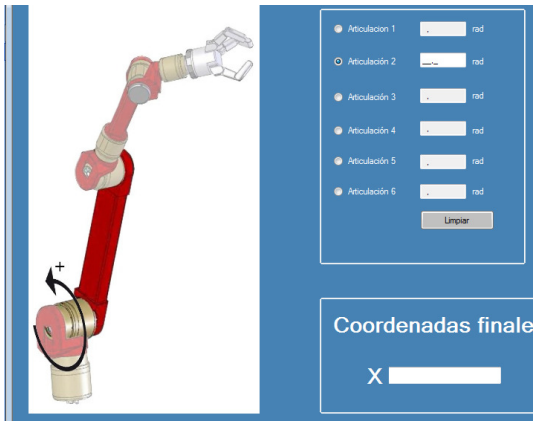


Ilustración 6.11: Módulo 2 seleccionado.

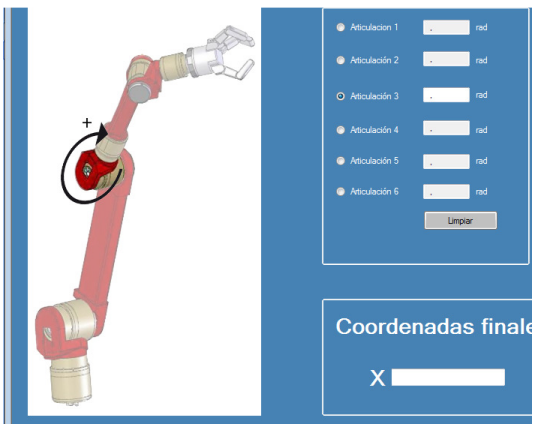


Ilustración 6.12: Módulo 3 seleccionado.

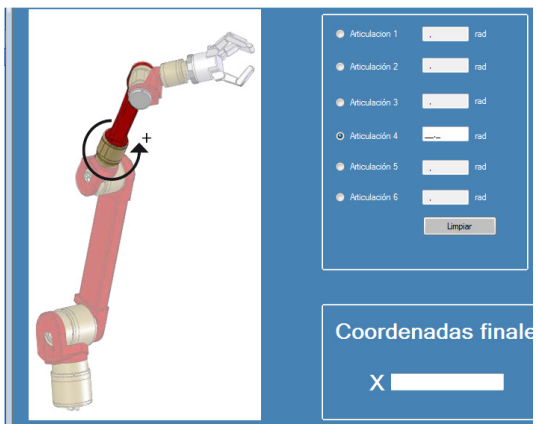


Ilustración 6.13: Módulo 4 seleccionado.

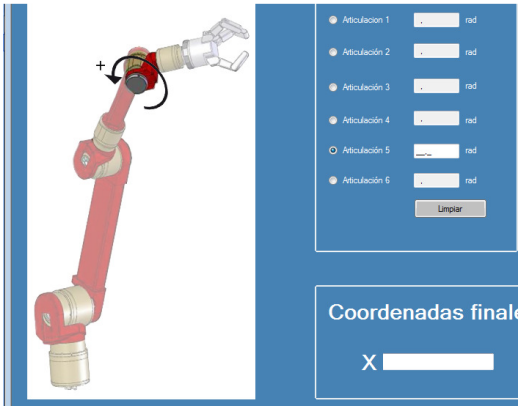


Ilustración 6.14: Módulo 5 seleccionado.

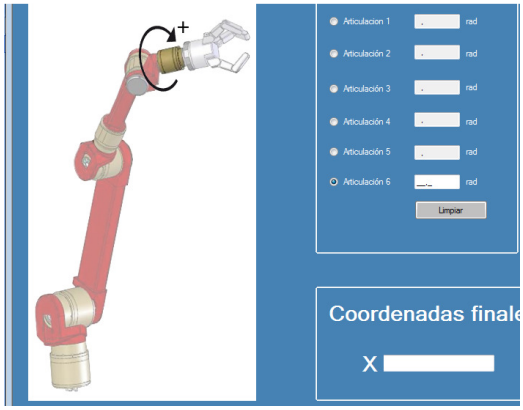


Ilustración 6.15: Módulo 6 seleccionado.

Existe la posibilidad de cargar automáticamente la posición Home y la Inicial, presionando sobre los botones correspondientes, de esta forma, se mostrarán los datos de en la pantalla con el objetivo de desplazar el robot a dichas posiciones como puede verse en la Ilustración 6.16, aunque también es posible modificar cualquiera de los mismos una vez cargados, siempre y cuando se actúe de la forma anteriormente citada.

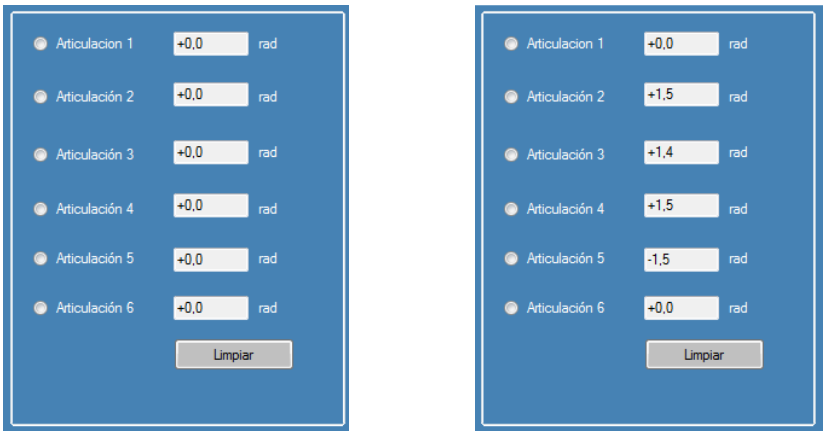


Ilustración 6.16: Seteo de la posición Home e Inicial respectivamente.

Adicionalmente existe un botón para Limpiar todos los datos de la pantalla, una seta para iniciar el movimiento y otra para realizar una parada de emergencia. Si se quiere manipular el robot es necesario que al menos una articulación tenga indicada una posición, sino se mostrará un mensaje de aviso alertando de que no hay información para realizar la operación (Ilustración 6.17). Además, si las posiciones indicadas no están dentro de los límites correspondientes también se avisará al usuario con el fin de que modifique la información, mientras no esté todo correcto no se podrá realizar el movimiento (Ilustración 6.18).

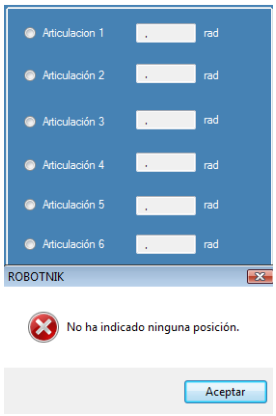


Ilustración 6.17: Mensaje error movimiento sin posiciones.

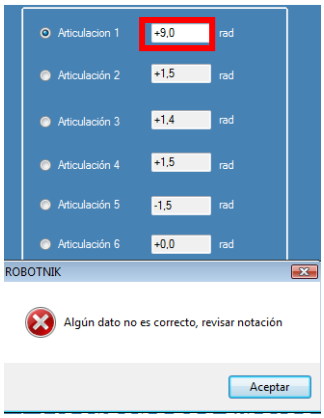


Ilustración 6.18: Mensaje error posiciones incorrectas.

Ilustración 6.19: Parada de emergencia activada.

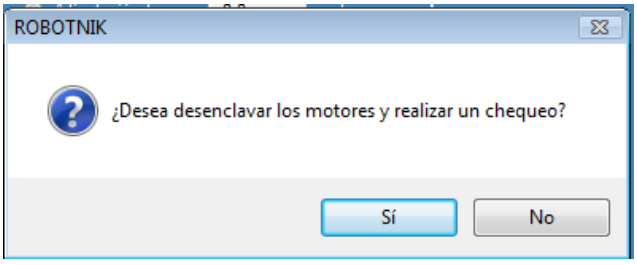


Ilustración 6.20: Autorización para desenclavamiento de motores y chequeo tras parada de emergencia.

La parada de emergencia puede pulsarse en cualquier momento, si estaba en movimiento se detendrán los motores que estuvieran actuando y bloqueará todos los módulos, si estaba parado, simplemente realizará el bloqueo. Para desenclavar los motores es necesario volver a pulsar la seta de emergencia, le saldrá el aviso mostrado en la Ilustración 6.20 al usuario para que confirme la operación y de ser así, realizará de nuevo la conexión con el dispositivo y un nuevo chequeo para comprobar que todos los módulos se han conectado correctamente y poder operar de nuevo con él, sino, mientras que persista algún error, quedará inhabilitada la teleoperación con el robot.

Por último, indicar que tras realizar un movimiento, o cuando se pulsa la parada de emergencia, se invoca a las funciones que calculan las coordenadas cartesianas del extremo del robot para indicarlas en los TextBox correspondientes que se encuentran en la parte inferior de la pantalla.

6.4. TELEOPERACIÓN MANUAL

La teleoperación manual consiste en ir controlando el movimiento del robot módulo a módulo realizando el control por posición o por velocidad (Ilustración 6.21). Antes de darle al botón "Avanzar" para que el robot se mueva, el usuario deberá haber seleccionado el motor, el control y el paso con el que desea trabajar. En el caso de realizar un control de posición deberá indicar el signo con las flechas correspondientes, en caso de un control de velocidad, deberá seleccionar el signo con los botones respectivos y usar las flechas para aumentar o disminuir la velocidad de trabajo.

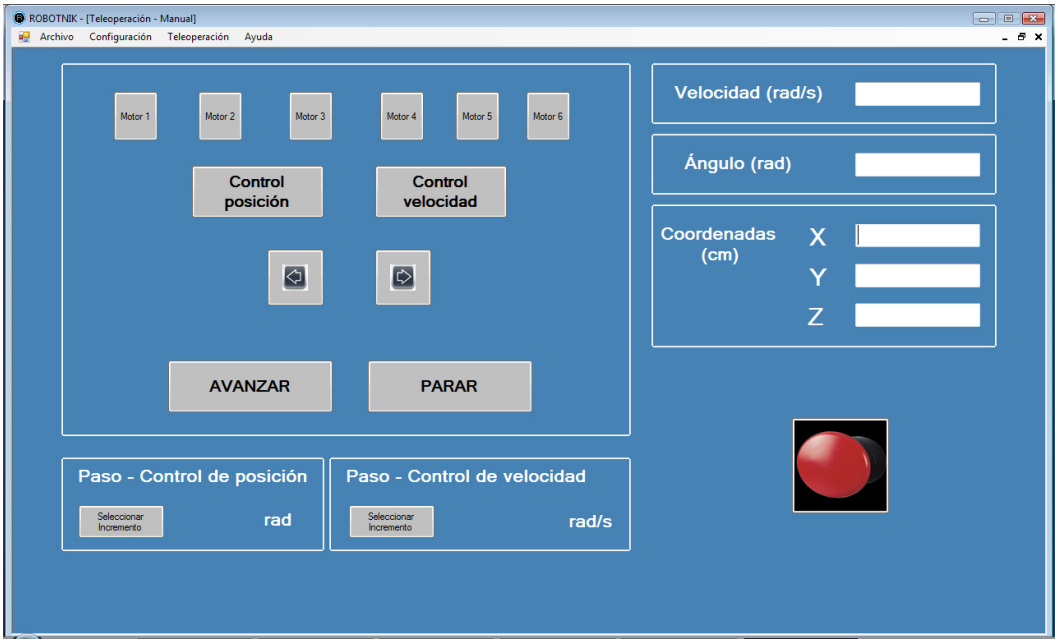


Ilustración 6.21: Ventana de teleoperación manual.

Si alguno de los datos requeridos no está indicado se mostrará el mensaje de aviso de la Ilustración 6.24 al usuario para que proceda a solventarlo y así poder manipular el robot.

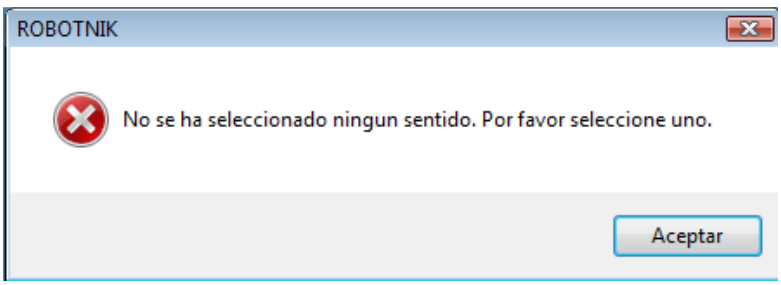


Ilustración 6.22: Mensaje error sentido no seleccionado.

Estos controles disponen de dos tipos de paradas, una suave que se realiza a través del botón "Parar", con la que sólo se detiene el módulo que se estuviera utilizando, y la parada de emergencia que funciona igual que en la teleoperación por ángulos, si se acciona, la seta se ilumina, se bloquean todos los motores y para desenclavarlos se debe volver a pulsar, se pedirá confirmación al usuario de la acción, si es afirmativa la seta se desactiva y se resetean todos los módulos sin excepción, si en este reseteo alguno de los módulos no ha vuelto a conectarse se avisará al usuario y no se podrá operar mientras no se solvente esta anomalía.

6.4.1. *Control de posición*

El control de posición se realiza para ir avanzando el paso seleccionado cada vez que se pulsa el botón "Avanzar", independientemente de la velocidad que tome el módulo indicado. Para ello, cada vez que se accede a esta opción de teleoperación, se actualizan las posiciones de los módulos, y algebraicamente se calcula la posición a la que tiene que llegar el módulo en cuestión, sumándole o restándole según el sentido, el paso indicado. En este control se emplea la función PCube_movePos(dev, i, θ_i), que como ya se ha indicado anteriormente, mueve el módulo i hacia la posición θ_i.

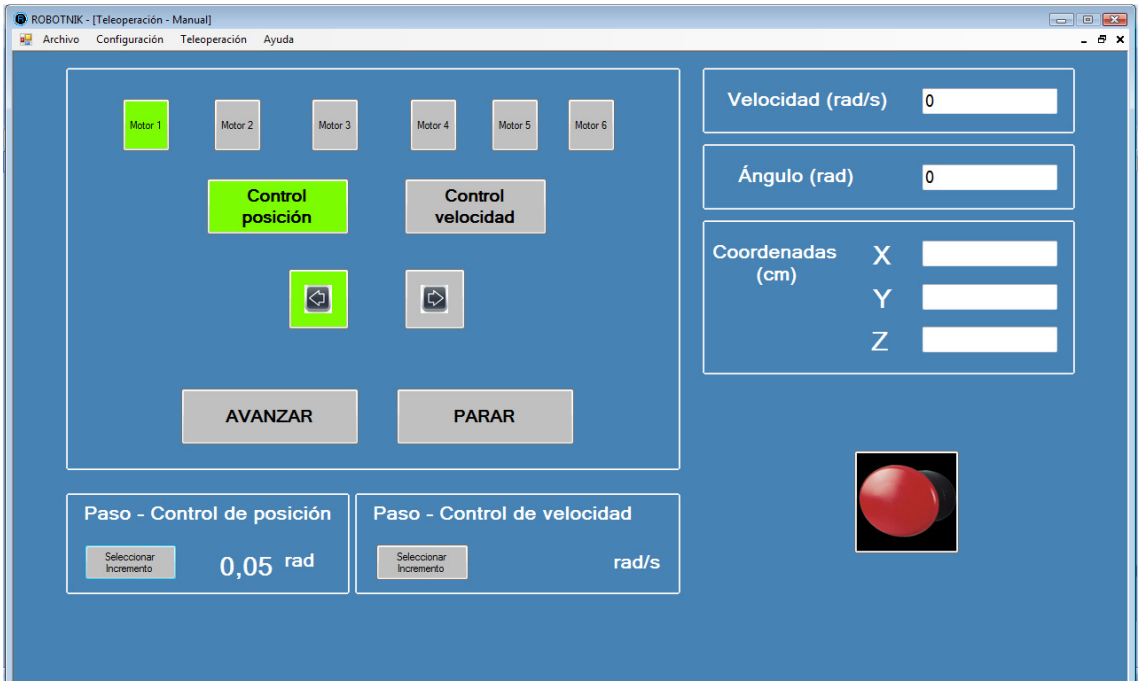


Ilustración 6.23: Configuración control posición.

Si al querer avanzar no se ha seleccionado ningún paso, se avisará al usuario con el mensaje de la Ilustración 6.24 para que lo elija entre las tres opciones que se definen en la configuración, Ilustración 6.25, que bien pueden ser los valores predeterminados u otros diferentes si es que el usuario ha realizado alguna modificación.

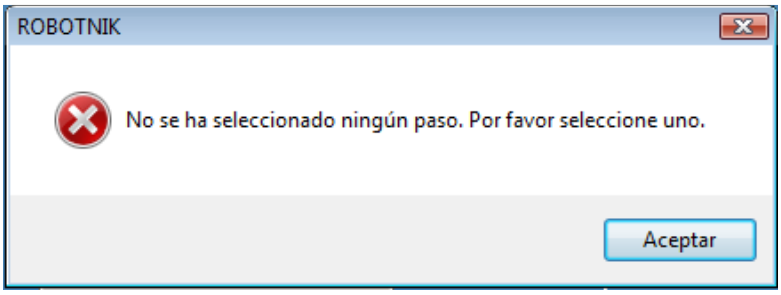


Ilustración 6.24: Mensaje error paso no seleccionado.

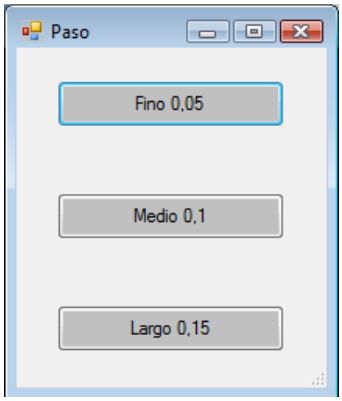


Ilustración 6.25: Ventana selección paso posición.

Si todo está correcto y el módulo elegido se desplaza hasta la posición correspondiente, se mostrarán las coordenadas cartesianas del extremo del robot en los TextBox diseñados a tal efecto, de igual forma, se indicará el ángulo en que se encuentra el módulo en cuestión.

Si con los sucesivos avances, la posición que se quiere alcanzar se encuentra fuera de los límites de operación del módulo en cuestión, se avisará al usuario de que el momento no puede realizarse con el mensaje de la Ilustración 6.26.

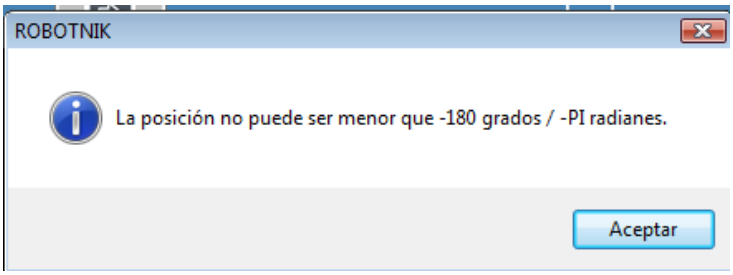


Ilustración 6.26: Mensaje aviso ángulo deseado fuera de los límites.

6.4.2. Control de velocidad

El control de velocidad está programado para hacer que el motor que se haya seleccionado se desplace en el sentido que se indique a una velocidad dada. Inicialmente la velocidad será de 0 rad/s, por lo que sólo se puede seleccionar la opción de aumentarla, de esta

forma, si se indica sentido positivo, aumento de velocidad, cada vez que se le dé al botón avanzar, el motor correspondiente se desplazará a velocidad anterior más el incremento seleccionado, de forma constante en el sentido que haya elegido el usuario. Si mientras que se está desplazando se cambia el sentido, se selecciona disminuir la velocidad, o se cambia el paso, al darle a "Avanzar" el motor reaccionará a tales cambios, cambiando su sentido de movimiento y disminuyendo/aumentando la velocidad en el paso correspondiente.

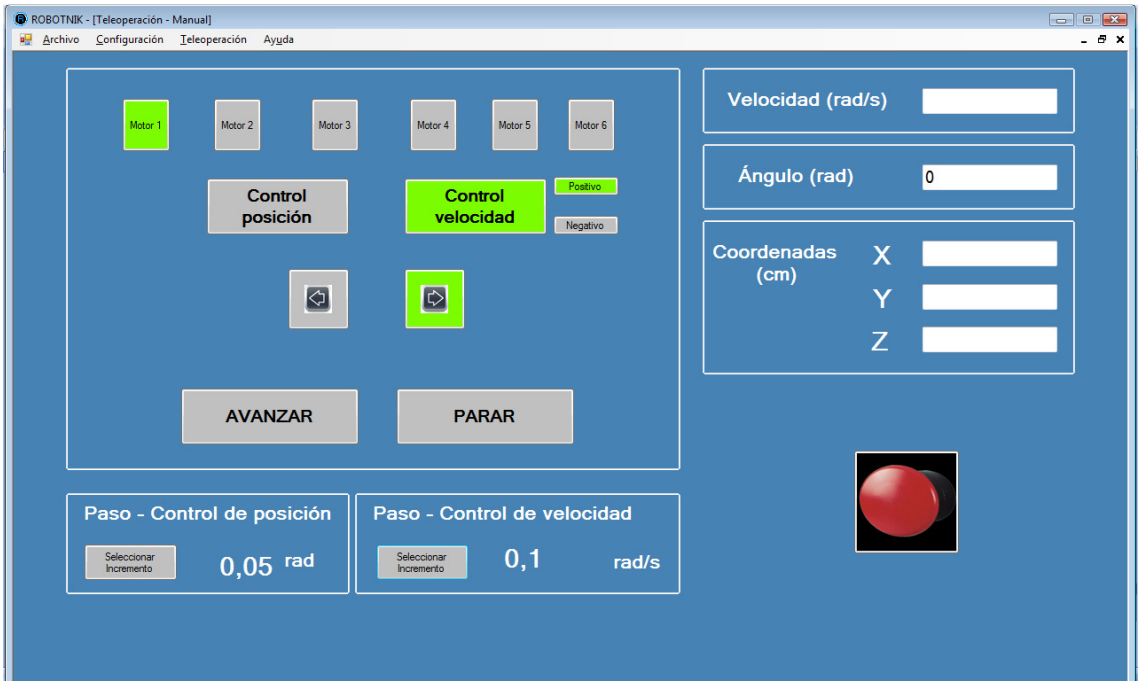


Ilustración 6.27: Control manual control de posición.

La velocidad máxima que se ha programado es de 10 rad/s, experimentalmente se ha comprobado que es una velocidad elevada, pero segura para el manejo del robot. Por otro lado la mínima se establece en 0 rad/s (Ilustración 6.28). En realidad la función de movimiento sí recibe la velocidad con signo, si es positiva se desplazará en su sentido positivo, y si es negativa en el contrario, sin embargo para resultar más intuitivo, la velocidad se aumenta o disminuye en valores absolutos y el signo se indica seleccionando una de las dos opciones habilitadas, realizando los cambios con la lógica programada.

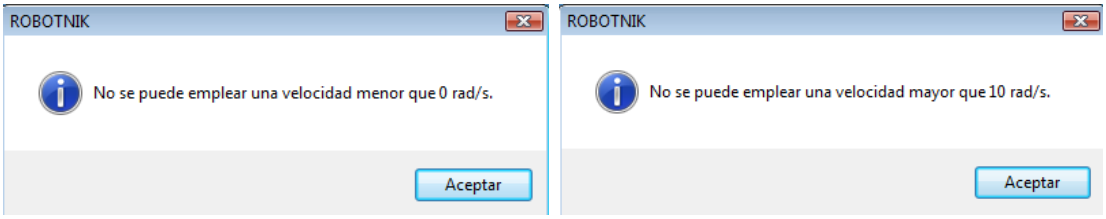


Ilustración 6.28: Mensajes aviso velocidad no permitida.

De igual forma que en el control por posición, si el paso no ha sido indicado, se mostrará un mensaje de aviso (Ilustración 6.29) y el usuario deberá seleccionar una de las tres

opciones habilitadas (Ilustración 6.30), siendo los valores los predefinidos si no los ha modificado el usuario, o en caso contrario, los que éste haya considerado.

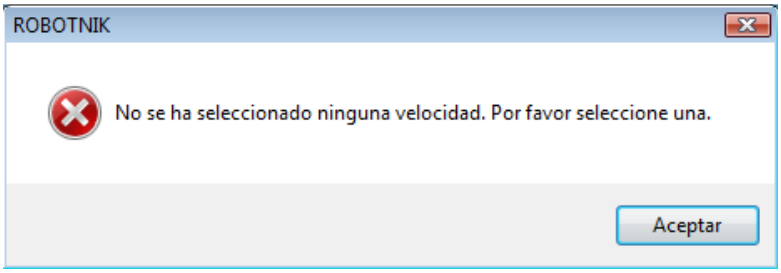


Ilustración 6.29: Mensaje error velocidad no seleccionada.

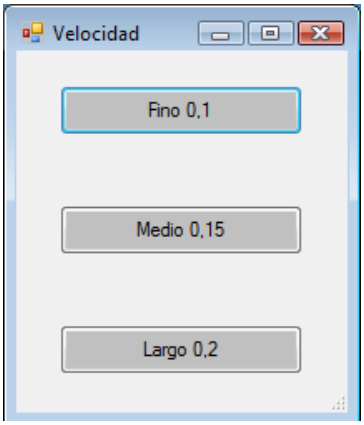


Ilustración 6.30: Ventana selección paso velocidad.

Con este tipo de control, es posible que los módulos lleguen a sus ángulos límites de trabajo y se bloqueen, en ese caso, el módulo se habrá parado y se advertirá al usuario de que no se puede utilizar el módulo puesto que se ha bloqueado (Ilustración 6.31), se le preguntará si desea desbloquearlo, si su respuesta es afirmativa, el motor se desenclavará y se avisará de nuevo al usuario que debe cambiar el sentido de movimiento si no quiere que se bloquee otra vez (Ilustración 6.33), si por el contrario la respuesta es negativa, el botón del motor bloqueado se pondrá de color rojo para recordar al usuario de que no está disponible (Ilustración 6.32), en este último caso, cuando el usuario quiera utilizar de nuevo este motor, se le solicitará de nuevo la autorización para desenclavarlo.

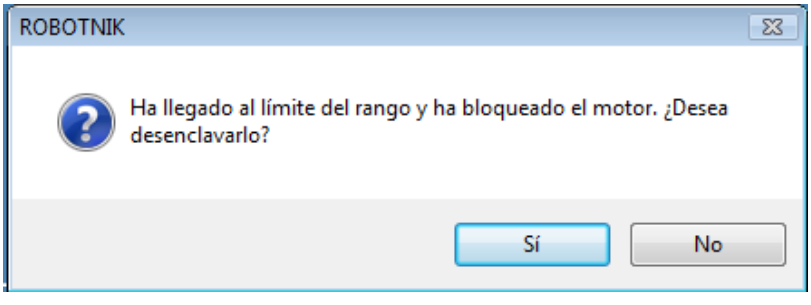


Ilustración 6.31: Mensaje aviso bloqueo de motor.

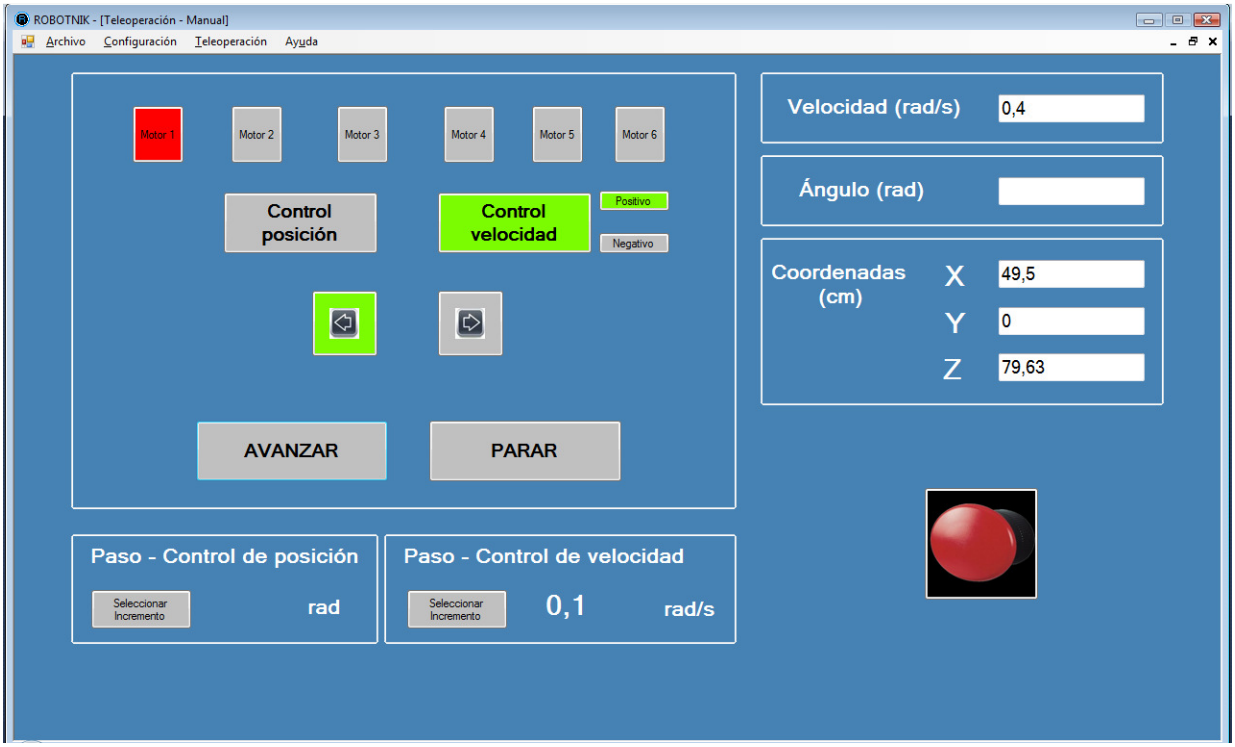


Ilustración 6.32: Teleoperación manual por control de velocidad con motor 1 bloqueado.

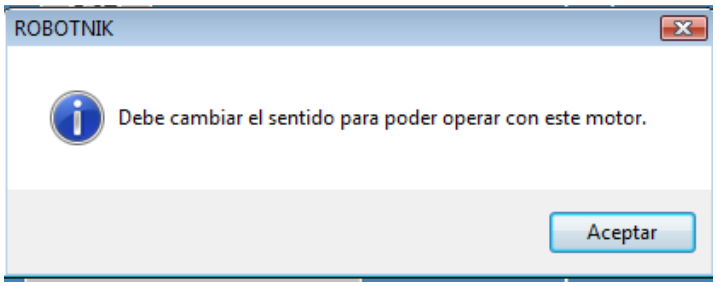


Ilustración 6.33: Mensaje aviso para cambiar el sentido de movimiento de un motor bloqueado.

6.5. AYUDA

El objetivo de este apartado de Ayuda es ofrecer al usuario información de interés acerca del propio robot, sus características físicas y técnicas, Así como información de las librerías empleadas en el desarrollo de la aplicación y detalles de la propia programación de la misma. Esto se realiza con el fin de facilitar en el futuro la ampliación de la aplicación, añadiendo otro tipo de funciones que se consideren oportunas, no sólo utilidades nuevas, sino también para incorporar las aplicaciones que otros estudiantes ya han diseñado para sus proyectos finales de carrera y de esta forma, ir creando una aplicación completa, interactiva y formativa con la que acercar el mundo de la robótica a los estudiantes y mostrar distintas aplicaciones prácticas realizadas por otros compañeros.

6.5.1. Robotnik

En este apartado de ayuda se muestran las especificaciones técnicas del robot, la hoja de características e información sobre los parámetros Denavit – Hartenberg. Con tres ventanas muy sencillas se tiene acceso a todos estos datos, donde queda recogida toda la información necesaria en caso de querer por ejemplo cambiar el robot de sitio, sustituir la fuente de alimentación, añadir o intercambiar módulos.

Ilustración 6.34: Características generales Robotnik.

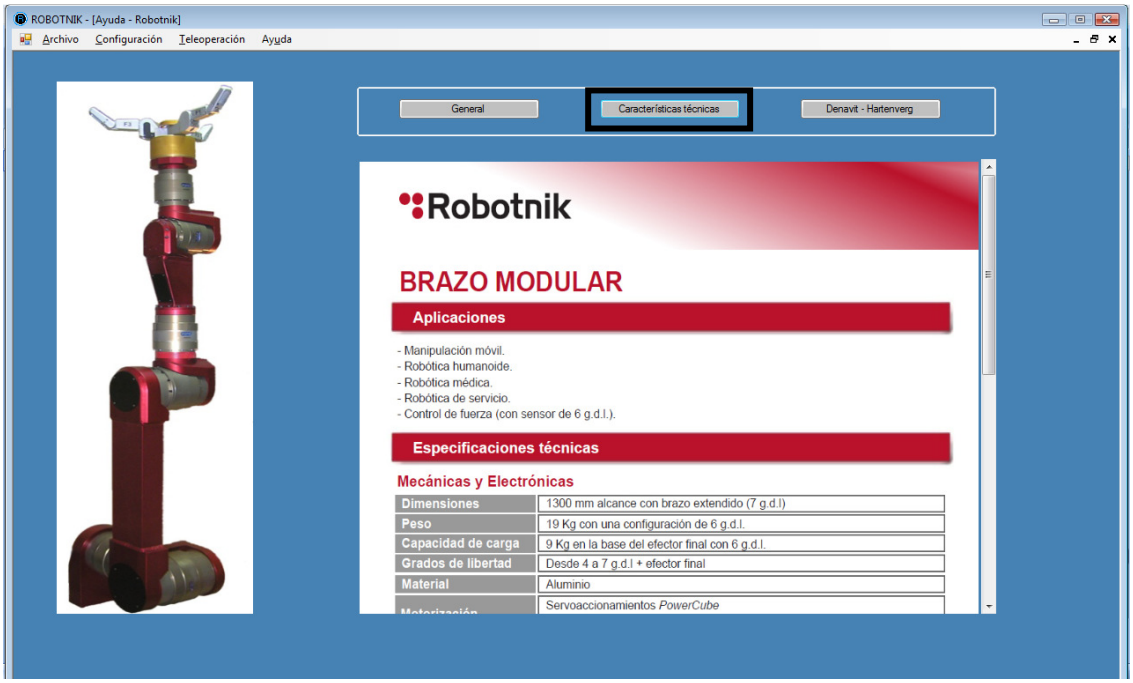


Ilustración 6.35: Hoja de características Robotnik.

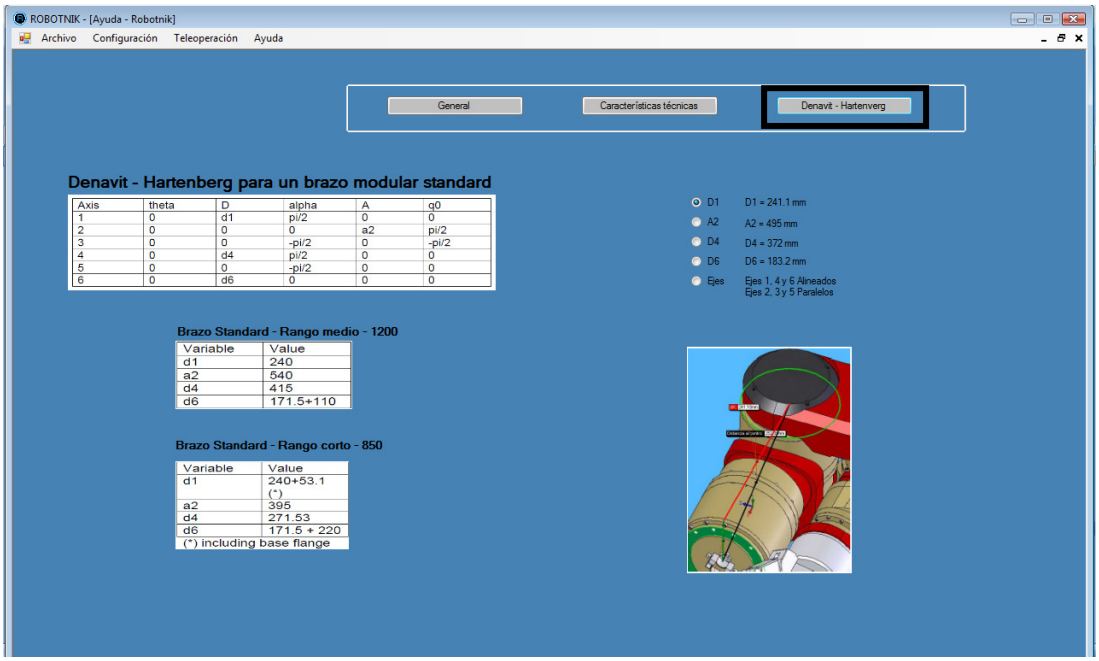


Ilustración 6.36: Información parámetros Denavit-Hartenberg.

6.5.2. Librerías

En cuanto a la ayuda sobre las librerías utilizadas, se detalla brevemente la funcionalidad de cada una de las empleadas, para que el usuario, en caso de querer ampliar la aplicación tenga conocimiento previo de las librerías de las que ya se dispone, sin tener que consultar el código.

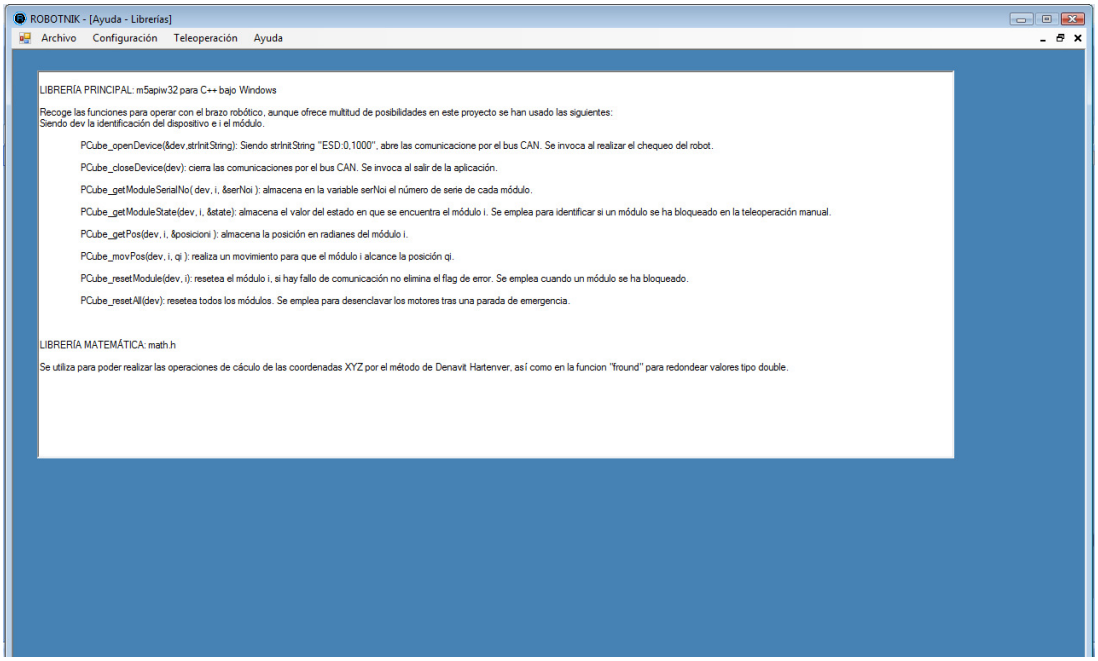


Ilustración 6.37: Ayuda Librerías.

6.5.3. Programación

Por último nos queda la ayuda de programación. Tiene el mismo objetivo que la ayuda de las librerías, evitar inicialmente que un usuario que quiera ampliar la aplicación tenga que consultar directamente el código para entender la forma de programación utilizada. Aunque el código se encuentre comentado, se ha considerado relevante el ofrecer información breve y concisa acerca de las funciones y variables empleadas, cual son sus funciones y el porqué de su uso. De esta forma, el usuario que quiera puede seguir ampliando esta interfaz siguiendo la misma filosofía para en un futuro, la aplicación resultante resulte un todo y no una suma de proyectos independientes, evitando incompatibilidades, posibles errores y confusión a otros programadores.

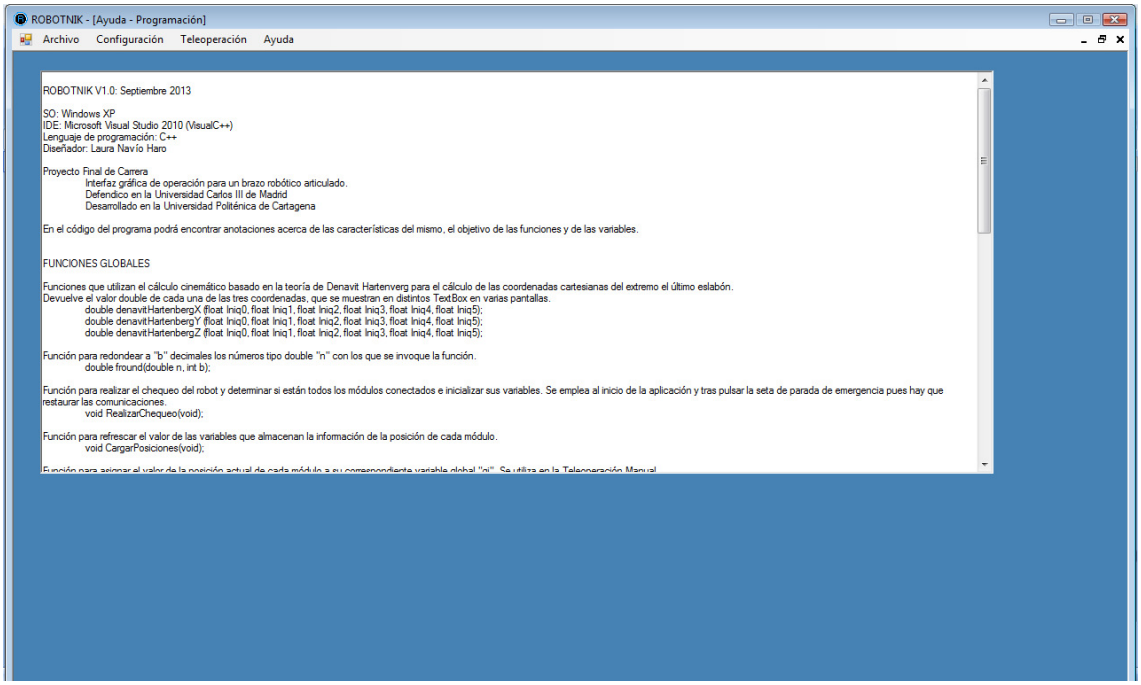


Ilustración 6.38: Ayuda programación.

CAPÍTULO 7

CONCLUSIONES Y MEJORAS

7.1 CONCLUSIONES

El objetivo formativo de este proyecto, como ya se ha dicho, ha sido la creación de una interfaz gráfica de manejo sencillo para la teleoperación de un brazo robótico articulado. Para ello se han valorado las opciones de programación existentes y se ha adoptado aquella en la que tenía cierta base de conocimientos. El entorno operativo Windows es el entorno sobre el que tengo mayor conocimiento, acerca del IDE utilizado destacar que a pesar de no haber trabajado nunca con él, se ofrecía como la mejor alternativa siendo un software muy completo y sobre el que hay multitud de referencias y bibliografía. Así mismo, aunque el lenguaje de programación elegido en un principio fue Visual Basic por lo intuitivo que resultaba a la hora desarrollar la interfaz, se cambió el desarrollo a C++ (Visual) debido a que con el primero hubo problemas de instalación y reconocimiento de la librería del robot. Con este segundo lenguaje, aunque la programación no es tan intuitiva, también hay suficientes referencias donde consultar, y con la práctica he alcanzado un nivel medio de conocimiento.

Además de adquirir nuevos conocimientos de programación, he podido poner en práctica por mi misma los conocimientos de robótica, manipulando un robot con distintas maneras de teleoperación, he adquirido la soltura para manejarme con librerías, funciones, he mejorado la lógica de programación, la metodología ensayo-error de tal manera que según avanzaba en el proyecto el tiempo de resolución de conflictos disminuía considerablemente.

Este proyecto ha sido un reto personal; afrontar de manera individual el planteamiento y desarrollo de una aplicación informática, sin conocimientos previos en muchos aspectos, pero con el fin de combinar dos ramas de tecnología en auge, la informática y la robótica.

7.2 MEJORAS

La interfaz creada es una base para posibles desarrollos de ampliaciones futuras, que pueden ser objeto de otros proyectos finales de carrera, o para la incorporación de algunos ya desarrollados. Como mejoras posibles podemos reseñar la implantación del control cinemático inverso; indicando las coordenadas cartesianas que se quieren alcanzar, calcular los parámetros necesarios para realizar el movimiento del robot. También se puede realizar un control dinámico, o la incorporación y adaptación de proyectos previos como el de Guillermo Martínez-Fortún Martínez; "Análisis del estado, configuración y puesta en marcha de un mecanismo de manipulación avanzado brazo-mano" consistente en la teleoperación de una mano articulada o el de Alejandro Sánchez Mur; "Control visual para el brazo Robotnik".

CAPÍTULO 8

PRESUPUESTO

8.1 COSTES DE MATERIAL

CONCEPTO	CANTIDAD	PRECIO UNITARIO	PRECIO TOTAL
SO WINDOWS VISTA HOME BASIC	1	299 €	299 €
IDE MS VISUAL STUDIO 2010 PROFESSIONAL	1	550 €	550 €
BRAZO ROBÓTICO	1	28000 €	28000 €
ORDENADOR	1	399 €	399 €
TARJETA CAN	1	45.95 €	45.95 €
		TOTAL	*29293.95 €*

8.2 COSTES DE PERSONAL

CONCEPTO	SUELDO MENSUAL BRUTO	MESES	SUELDO TOTAL
INGENIERO INDUSTRIAL	2.000€	6	12.000€
		TOTAL BRUTOS	12.000€
		TOTAL NETOS (-18%)	*9.841€*

8.3 COSTE TOTAL

CONCEPTO	*TOTAL*
COSTE MATERIAL	29293.95 €
COSTE PERSONAL	9.841 €
TOTAL	*38774.95€*

CAPÍTULO 9

BIBLIOGRAFÍA

9.1 RECURSOS ELECTRÓNICOS PRINCIPALES

[1]; http://buscon.rae.es/drae/?type=3&val=robotica&val_aux=&origen=REDRAE

[2]; http://www.roboticspot.com/robotica.php

[3]; http://robotec11.tripod.com/

[5]; http://www.wordreference.com/definicion/aut%C3%B3mata

[6]; http://buscon.rae.es/drae/?type=3&val=AUTOMATA&val_aux=&origen=REDRAE

[7]; http://buscon.rae.es/drae/?type=3&val=robot&val_aux=&origen=REDRAE

[8]; http://enciclopedia.us.es/index.php/Tres_leyes_de_la_rob%C3%B3tica

[9]; http://www.amazings.com/ciencia/noticias/100709d.html

[10]; http://www.robotikka.com/4398/robots/submarinos-roboticos-buscaran-naufragos-en-baja-california/

[11]; http://www.is.sys.es.osaka-u.ac.jp/index.en.html

[12]; http://www.profesormolina.com.ar/tecnologia/robotica/historia.htm

[13]; http://es.scribd.com/doc/49615065/Tipos-de-robots-industriales

[14]; http://catarina.udlap.mx/u_dl_a/tales/documentos/msp/de_l_ea/apendiceB.pdf

[15]; http://www.ciencia-explicada.com/2013/02/parametrizacion-denavit-hartenberg-para.html

[16]; http://es.wikipedia.org/wiki/Microsoft_Windows

[17]; http://es.wikipedia.org/wiki/GNU/Linux

[18]; http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n

[19]; http://www.biografiasyvidas.com/biografia/n/neumann.htm

[20]; http://h2non.wordpress.com/2007/11/07/buses-de-comunicacion-von-neumann/

[21]; http://canbus.galeon.com/electronica/canbus.htm

[22]; http://es.scribd.com/doc/7305395/CAN-Bus-Bosch

[23]; http://wiki.webdearde.com/index.php/Grados_de_libertad

9.2 RECURSOS BIBLIOGRÁFICOS

[4]; Tema 2, Automatización Industrial II. 2º Cuatrimestre, 3º Curso I.T.I. E.I.

9.2.1 Manuales

- ROBOTNIK_MODULAR_ARM-Installation and System Configuration Manual.pdf
- ROBOTNIK_MODULAR_ARM-Robot Dynamics Manual.pdf
- ROBOTNIK_MODULAR_ARM-Robot kinematics and control.pdf
- ROBOTNIK_MODULAR_ARM-System Architecture Manual.pdf
- ROBOTNIK_MODULAR_ARM-System Startup Manual.pdf

CAPÍTULO 10

ANEXOS

- ANEXO I. Funciones librería m5apiw32.
- ANEXO II. Cinemática del robot(Extracto manual "*Robot kinematics and control*").
- ANEXO III. Informe reflejado en el archivo 'Chequeo.txt'.
- ANEXO IV. Hojas de características.

ANEXO I. Funciones librería m5apiw32.

Opening and closing the communication Interface	Function	Description
	PCube_openDevice	Opens the interface by specifiing an InitString. Result is a valid deviceID. Soo documont "First stops" for furthor information on tho InitString.
	PCube_closeDevice	Closes the interface by specifiing the deviceID.

Administrative functions	Function	Description
	PCube_getModuleIdMap	Retrieves the number of PowerCube modules found on the bus. At the same time this function maps the physical addresses of the modules to logical IDs. The physical addresses are strored in an ascending order in the array specified.
	PCube_updateModuleIdMap	Redoes the mapping.
	PCube_getModuleCount	Retrieves the number of modules connected to the bus.
	PCube_getModuleType	Retrieves the module type by specifiing deviceID and a moduleID. Rotary drives: TYPEID_MOD_ROTARY = 0x0F Linear drives: TYPEID_MOD_LINEAR = 0xF0
	PCube_getModuleVersion	Retrieves the version of the operating system of the module by specifiing deviceID and a moduleID. The result has to be interpreted as a hexadecimal number.
	PCube_getModuleSerialNo	Retrieves the serial number of a module by specifiing deviceID and a moduleID.
	PCube_getDllVersion	Retrieves the version of the running DLL.
	PCube_configFromFile	Configures the complete system to the specifications in a Ini file. The file name is a parameter for the call. See section "System configuration using an Ini file".
	PCube_serveWatchdogAll	Refreshs the watchdog in all conected modules if these are enabled. Valid only for CAN bus.
	PCube_getDefSetup	Retrieves the default module setup by specifiing deviceID and a moduleID. Result is a setup word, separatly described in the section "Module configuration".
	PCube_getDefBaudRate	Retrieves the default Baudrate of the module (useful only for CAN- and RS232 modules). Result is a value between 0..5. CAN: 0=50 1=250 2=500 4=1000 kbit/s RS232: 0=1200 1=2400 2=4800 3=9600 4=19200 5=38400 bit/s
	PCube_setBaudRateAll	Broadcast command to change the baudrate of all connected modules (only CAN bus). All modules conected immediately change their baudrate to the new value. Only a valid baudrate will be accepted. Valid only for CAN-Bus.
	PCube_getDefGearRatio	Retrieves the default gear ratio.
	PCube_getDefLinearRatio	Retrieves the default factor for conversion of rotary to linear motion.
	PCube_getDefCurRatio	Retrieves the default factor for conversion of current digits to A.
	PCube_getDefBrakeTimeOut	Retrieves the default delay between end of motion and release of the brake.
	PCube_getDefIncPerTurn	Retrieves the default value for the number of increments per motor rotation.

Retrieve position	Function	Description
	PCube_getPos	Retrieves the actual module position by specifiing deviceID and a moduleID. Result is the position in rad (rotary) or m (linear modules).
	PCube_getPosInc	Retrieves the actual module position by specifiing deviceID and a moduleID. Result is the position in increments.
	PCube_getPosCountInc	Retrieves the current counter value by specifiing deviceID and a moduleID. Result is the current counter value in increments (position without any offsets).

Retrieve speed	Function	Description
	PCube_getVel	Retrieves the current speed by specifiing deviceID and moduleID. Result is the real speed in rad/s (rotary) or m/s (linear modules).
	PCube_getVelInc	Retrieves the current speed by specifiing deviceID and moduleID. Result is the real speed in increments/s.
	PCube_getIPolVel	Retrieves the current interpolated speed by specifiing deviceID and moduleID. Result is the interpolated speed in rad/s (rotary) or m/s (linear modules).

Retrieve current	Function	Description
	PCube_getCur	Retrieves the actual current information by specifiing deviceID an a moduleID. Result is the actual current in A.
	PCube_getCurInc	Retrieves the actual current information by specifiing deviceID an a moduleID. Result is the actual current in Digits.

Retrieve tow distance	Function	Description
	PCube_getDeltaPos	Retrieves the actual tow distance by specifiing deviceID and a moduleID. Result is the actual tow distance in rad (rotary) or m (linear modules).
	PCube_getDeltaPosInc	Retrieves the actual tow distance by specifiing deviceID and a moduleID. Result is the actual tow distance in increments.

Retrieve module state	Function	Description
	PCube_getModuleState	Retrieves the actual module state by specifiing deviceID and a moduleID. Result is the module status word, separately described in section „Module state".
	PCube_getStateDioPos	Retrieves a combined information on module state, position and digital IO state. Result is the module state (section Module state), the actual position in rad resp. m and the state of the digital I/Os.

Retrieve position synchronously	Function	Description
	PCube_savePosAll	This broadcast command forces all connected modules to save their current position at the same time. The deviceID is a necessary parameter. For CAN Bus only.
	PCube_getSavePos	Retrieves the position value saved during the call PCube_savePosAll by specifiing deviceID and a moduleID. Result is the saved with PCube_savePosAll position in rad resp. m. For CAN-Bus only.

Configuration	Function	Description
	PCube_getDefConfig	Retrieves the default module configuration by specifiing deviceID and a moduleID. Result is the configuration word, separately described in section „Module configuration".
	PCube_getConfig	Retrieves the actual module configuration by specifiing deviceID and a moduleID. Result is the configuration word as saved to the module with the last call of PCube_setConfig. After Power on this value is identical to the default value.
	PCube_setConfig	Sets the actual module configuration by specifiing deviceID, a moduleID and a new configuration word.

Digitale I/O	Function	Description
	PCube_getDefDioData	Retrieves the default state of the digital IOs by specifiing deviceID and a moduleID. Result is the IO state described in section "Module state".
	PCube_getDioData	Retrieves the actual state of the digital IOs by specifiing deviceID and a moduleID. Result is the actual IO state

Digitale I/O	Function	Description
	PCube_setDioData	Sets the actual IO state (only outputs) by specifiing deviceID, a moduleID and a valid IO state word.

PID loop coefficients	Function	Description
	PCube_getDefA0	Retrieves the default value of the PID loop coefficient A0.
	PCube_getA0	Retrieves the actual value of the PID loop coefficient A0. After Power on this value is identical to the default.
	PCube_setA0	Sets the actual value of the PID loop coefficient A0 (range 1..12)
	PCube_getDefC0	Retrieves the default value of the PID loop coefficient C0.
	PCube_getC0	Retrieves the actual value of the PID loop coefficient C0. After Power on this value is identical to the default.
	PCube_setC0	Sets the actual value of the PID loop coefficient C0 (range 12..64, even values only)
	PCube_getDefDamp	Retrieves the default value of the PID loop coefficient "Damping".
	PCube_getDamp	Retrieves the actual value of the PID loop coefficient "Damping". After Power on this value is identisch to the default.
	PCube_setDamp	Sets the actual value of the PID loop coefficient "Damping" (range 1..4)
	PCube_recalcPIDParams	Call to update the PID loop and to make the new coefficients valid. This function must be called after A0, C0 or Damp have been altered.

Position offset	Function	Description
	PCube_getDefHomeOffset	Retrieves the default Home offset by specifiing deviceID and a moduleID. Result is the default home offset in rad resp. m. The home offset is the position value in home position.
	PCubo_gotHomoOffsot	Rotriovos tho actual Homo offsot by spocifiing dovicoID and a moduloID. Result is the actual home offset in rad resp. m. After Power on this value is identical to the default.
	PCubo_gotHomoOffsotInc	Rotriovos tho actual Homo offsot by spocifiing dovicoID and a moduloID. Result is the actual home offset in Increments.
	PCube_setHomeOffset	Sets the actual Home offset by specifiing deviceID, a moduleID and the new value in rad resp. m.
	PCube_setHomeOffsetInc	Sets the actual Home offset by specifiing deviceID, a moduleID and the new value in Increments.

Homing speed	Function	Description
	PCube_getDefHomeVel	Retrieves the default Homing speed by specifiing deviceID and a moduleID. Result is the default Homing speed in rad/s resp. m/s. This speed is used during the homing procedure.
	PCube_getHomeVel	Retrieves the actual Homing speed by specifiing deviceID and a moduleID. Result is the Homing speed in rad/s resp. m/s. After Power on this value is identical to the default.
	PCube_getHomeVelInc	Retrieves the actual Homing speed by specifiing deviceID and a moduleID. Result is the Homing speed in Increments/s.
	PCube_setHomeVel	Sets the actual Homing speed by specifiing deviceID, a moduleID and the new value in rad/s resp. m/s.
	PCube_setHomeVelInc	Sets the actual Homing speed by specifiing deviceID, a moduleID and the new value in Increments/s.

Operation range: Minimum position	Function	Description
	PCube_getDefMinPos	Retrieves the default minimum position by specifiing deviceID and a moduleID. Result is the minimum position in rad resp. m. This parameter is used as a limit for the operation range. Values less than this will be limited to the given minimum.
	PCube_getMinPos	Retrieves the actual minimum position by specifiing deviceID and a moduleID. Result is the minimum position in rad resp. m. After Power on this value is identical to the default.
	PCube_getMinPosInc	Retrieves the actual minimum position by specifiing deviceID and a moduleID. Result is the minimum position in increments.
	PCube_setMinPos	Sets the actual minimum position by specifiing deviceID, a moduleID and the new value in rad resp. m.

Operation range: Minimum position	Function	Description
	PCube_setMinPosInc	Sets the actual minimum position by specifiing deviceID, a moduleID and the new value in Increments.

Operation range: Maximum position	Function	Description
	PCube_getDefMaxPos	Retrieves the default maximum position by specifiing deviceID and a moduleID. Result is the maximum position in rad resp. m. This parameter is used as a limit for the operation range. Values greater than this will be limited to the given maximum.
	PCube_getMaxPos	Retrieves the actual maximum position by specifiing deviceID and a moduleID. Result is the maximum position in rad resp. m. After Power on this value is identical to the default.
	PCube_getMaxPosInc	Retrieves the actual maximum position by specifiing deviceID and a moduleID. Result is the maximum position in increments.
	PCube_setMaxPos	Sets the actual maximum position by specifiing deviceID, a moduleID and the new value in rad resp. m.
	PCube_setMaxPosInc	Sets the actual maximum position by specifiing deviceID, a moduleID and the new value in Increments.

Maximum speed	Function	Description
	PCube_getDefMaxVel	Retrieves the default maximum speed by specifiing deviceID and a moduleID. Result is the maximum speed in rad/s resp. m/s. This parameter is used as a limit. Values greater than this will be limited to the maximum.
	PCube_getMaxVel	Retrieves the actual maximum speed by specifiing deviceID and a moduleID. Result is the maximum speed in rad/s resp. m/s. After Power on this value is identical to the default.
	PCube_getMaxVelInc	Retrieves the actual maximum speed by specifiing deviceID and a moduleID. Result is the maximum speed in Increments/s.
	PCube_setMaxVel	Sets the maximum speed by specifiing deviceID, a moduleID and the new value in rad/s resp. m/s.
	PCube_setMaxVelInc	Sets the maximum speed by specifiing deviceID, a moduleID and the new value in Increments/s.

Maximum acceleration	Function	Description
	PCube_getDefMaxAcc	Retrieves the default maximum acceleration by specifiing deviceID and a moduleID. Result is the maximum acceleration in rad/s² resp. m/s². This parameter is used as a limit. Values greater than this will be limited to the maximum.
	PCube_getMaxAcc	Retrieves the actual maximum acceleration by specifiing deviceID and a moduleID. Result is the maximum acceleration in rad/s² resp. m/s². After Power on this value is identical to the default.
	PCube_getMaxAccInc	Retrieves the actual maximum acceleration by specifiing deviceID and a moduleID. Result is the maximum acceleration in Increments/s².
	PCube_setMaxAcc	Sets the maximum speed by specifiing deviceID, a moduleID and the new value in rad/s² resp. m/s².
	PCube_setMaxAccInc	Sets the maximum speed by specifiing deviceID, a moduleID and the new value in Increments/s².

Maximum current	Function	Description
	PCube_getDefMaxCur	Retrieves the default maximum current by specifiing deviceID and a moduleID. Result is the maximum current in A. This value is a limit for the maximum motor current used during operation.
	PCube_getMaxCur	Retrieves the actual maximum current by specifiing deviceID and a moduleID. Result is the maximum current in A. After Power on this value is identical to the default.
	PCube_setMaxCur	Sets the actual maximum current by specifiing deviceID, a moduleID the new maximum in A.

Ramp motion with specification of Target position	Function	Description
	PCube_movePos	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m. Prior to this call target speed and acceleration must be set using Funktionen PCube_setRampVel and PCube_setRampAcc resp. PCube_setRampVelInc and PCube_setRampAccInc.
	PCube_movePosExtended	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m. Results of this call are State, actual position and Digital IO state (like PCube_getStatePosDio).

Ramp motion with specification of position, speed and acceleration	Function	Description
	PCube_moveRamp	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m, target speed in rad/s resp. m/s and target acceleration in rad/s² resp. m/s².
	PCube_moveRampInc	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in Increments, target speed in Increments/s and target acceleration in Increments/s².
	PCube_moveRampExtended	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m, target speed in rad/s resp. m/s and target acceleration in rad/s² resp. m/s². Results of this call are State, actual position and Digital IO state (like PCube_getStatePosDio).

Constant speed motion	Function	Description
	PCube_moveVel	Starts a constant speed motion. Target speed is specified in rad/s resp. m/s.
	PCube_moveVelInc	Starts a constant speed motion. Target speed is specified in Increments/s.
	PCube_moveVelExtended	Starts a constant speed motion. Target speed is specified in rad/s resp. m/s. Results of this call are State, actual position and Digital IO state (like PCube_getStatePosDio).

Constant current motion	Function	Description
	PCube_moveCur	Starts a constant current motion. The target currrent is specified in A.
	PCube_moveCurInc	Starts a constant current motion. The target currrent is specified in Digits.
	PCube_moveCurExtended	Starts a constant current motion. The target currrent is specified in A. Results of this call are State, actual position and Digital IO state (like PCube_getStatePosDio).

Motion with specification of target position and time	Function	Description
	PCube_moveStep	Starts motion to the target position specified in rad resp. m. Target time for the ride is specified in ms.
	PCube_moveStepInc	Starts motion to the target position specified in increments. Target time for the ride is specified in ms.
	PCube_moveStepExtended	Starts motion to the target position specified in rad resp. m. Target time for the ride is specified in ms. Results of this call are State, actual position and Digital IO state (like PCube_getStatePosDio).

Synchronisierter Start aller Antriebe mit Zielvorgabe	Function	Description
	PCube_startMotionAll	If the configuration of all connected modules has been altered (see module configuration) a synchronous motion command can be issued. By sending PCube_startMotionAll all connected modules start their motion command at exactly the same time. For CAN-Bus only.

Function	Description	Description
	PCube_initEMS_IO	Initializes an EMS module on the bus specified by deviceID, Type and Serial number. Result of this call is a valid moduleID for the chosen EMS module. For CAN-Bus only.
	PCube_getDataEMS_DIO	Retrieves data from the EMS module specified by deviceID and moduleID. Result is the state of the channel specified by channelID. The moduleID must have been requested prior to this using PCube_initEMS_IO. For CAN-Bus only. For Digital EMS IO-Moduls only.
	PCube_getDataEMS_AIO	Retrieves data from the EMS module specified by deviceID and moduleID. Result is the value of the channel specified by channelID. The moduleID must have been requested prior to this using PCube_initEMS_IO. For CAN-Bus only. For Analog EMS IO-Moduls only.
	PCube_setDataEMS_DIO	Sets data on the EMS module specified by deviceID and moduleID. The state given is transferred to the channel chosen with channelID. For CAN-Bus only. For Digital EMS IO-Moduls only.
	PCube_setDataEMS_AIO	Sets data on the EMS module specified by deviceID and moduleID. The value given is transferred to the channel chosen with channelID. For CAN-Bus only. For Analog EMS IO-Moduls only.

Retrieve data from DLR Force torque sensor	Function	Description
	PCube_initDLR_FTS	Initializes the Force Ttorque sensor on the bus by specifiing the deviceID. For CAN-Bus only. Only one Sensor per device (bus) is allowed.
	PCube_getDataDLR_FTS	Retrieves data from the Force Torque sensor. Results are 3 force values (X,Y,Z) and 3 torque values (X,Y,Z) as well as the sensor state. For CAN-Bus only.

Retrieve data from SCHUNK Force torque sensor	Function	Description
	PCube_getDataSCHUNK_FTC	Retrieves data from the Force Torque sensor. Results are 3 force values (X,Y,Z) and 3 torque values (X,Y,Z) or 3 translation values (X,Y,Z) and 3 rotation values (X,Y,Z) as well as the sensor state. For CAN-Bus only.
	PCube_setNullSCHUNK_FTC	Nulls the Force Torque sensor. Results is the sensor state. For CAN-Bus only.

Retrieve data from MP55, produced by HBM	Function	Description
	PCube_getDataMP55_IO	Retrieves the actual measurement data from a MP55 specified by deviceID and moduleID. For CAN-Bus only.
	PCube_setTaraMP55_IO	Tares the MP55 specified by deviceID and moduleID. For CAN Bus only.

Maximum tow distance	Function	Description
	PCube_getDefMaxDeltaPos	Retrieves the default maximum tow distance by specifiing deviceID and a moduleID. Result is the default maximum tow distance in rad resp. m. This is a limiting value for the maximum tow distance allowed. If the drive overshoots this value during operation an error will be generated and the motor stops.
	PCube_getMaxDeltaPos	Retrieves the actual maximum tow distance by specifiing deviceID and a moduleID. Result is the actual maximum tow distance in rad resp. m. After Power on this value is identical to the default.
	PCube_getMaxDeltaPosInc	Retrieves the actual maximum tow distance by specifiing deviceID and a moduleID. Result is the actual maximum tow distance in increments.
	PCube_setMaxDeltaPos	Sets the actual maximum tow distance by specifiing deviceID, a moduleID and the new maximum in rad resp. m.
	PCube_setMaxDeltaPosInc	Sets the actual maximum tow distance by specifiing deviceID, a moduleID and the new maximum in Increments.

Target ramp motion speed	Function	Description
	PCube_setRampVel	Sets the target speed for a ramp motion profile by specifiing deviceID, a moduleID and the new value in rad/s resp. m/s. This value will be used for all ramp motion commands started with PCube_movePos, PCube_movePosInc or PCube_movePosExtended. In order to do so a target acceleration greater than zero must have been set using PCube_setRampAcc or PCube_setRampAccInc.
	PCube_setRampVelInc	Sets the target speed for ramp motion profiles by specifiing deviceID, a moduleID and the new value in Increments/s.

Target ramp motion acceleration	Function	Description
	PCube_setRampAcc	Sets the target acceleration for a ramp motion profile by specifiing deviceID, a moduleID and the new value in rad/s² resp. m/s². This value will be used for all ramp motion commands started with PCube_movePos, PCube_movePosInc or PCube_movePosExtended. In order to do so a target speed greater than zero must have been set using PCube_setRampVel oder PCube_setRampVelInc.
	PCube_setRampAccInc	Sets the target acceleration for ramp motion profiles by specifiing deviceID, a moduleID and the new value in Increments/s².

Homing	Function	Description
	PCube_homeModule	Starts a Homing procedure of the module specified by deviceID and moduleID.
	PCube_homeAll	Starts a Homing procedure of all modules connected to the bus. For CAN-Bus only.

Quick stop	Function	Description
	PCube_haltModule	Issues a Quick stop of the module specified by deviceID and moduleID.
	PCube_haltAll	Issues a Quick stop of all modules connected to the bus. For CAN-Bus only.

Reset of module state	Function	Description
	PCube_resetModule	Issues a Reset of the module specified by deviceID and moduleID. A Reset can clear error flags in the module state. If an error is permanent, Reset is ignored.
	PCube_resetAll	Issues a Reset of all modules connected to the bus. For CAN-Bus only.

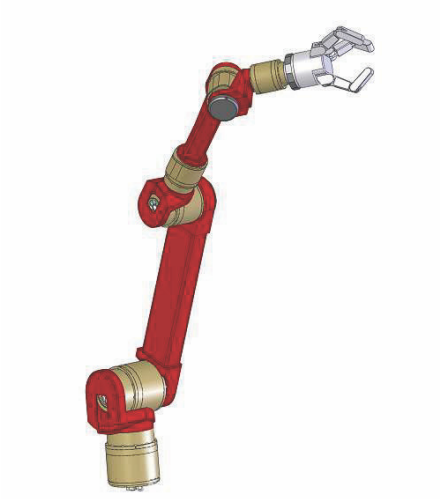
Ramp motion with specification of Target position	Function	Description
	PCube_movePos	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in rad resp. m. Prior to this call target speed and acceleration must be set using Funktionen PCube_setRampVel and PCube_setRampAcc resp. PCube_setRampVelInc and PCube_setRampAccInc.
	PCube_movePosInc	Starts a ramp motion profile of the module specified by deviceID and moduleID. The target position is given in increments.

ANEXO II. Cinemática del robot (Extracto manual "*Robot kinematics and control*").

1. Robot kinematics

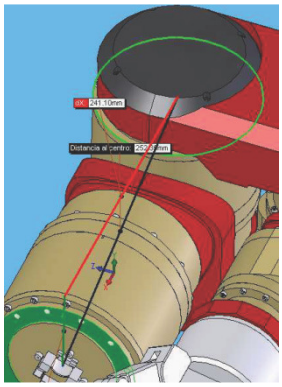
1.1 Denavit Hartenberg parameters

The Standard Modular Robot Arm configuration has 6DOF, RRR for position and a RPR (Roll-Pitch-Roll) wrist. The 6DOF configuration is equivalent to the well known Puma manipulator.

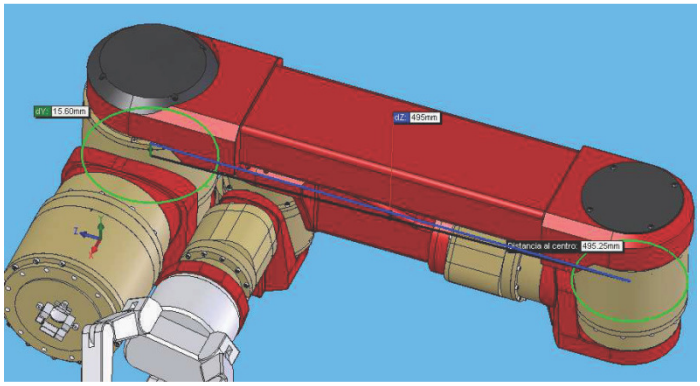


The Denavit-Hartenberg parameters of the standard arm (length 1551mm) are:

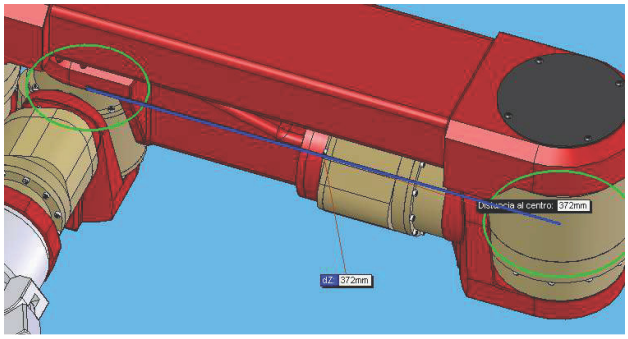
D1 = 241.1 mm



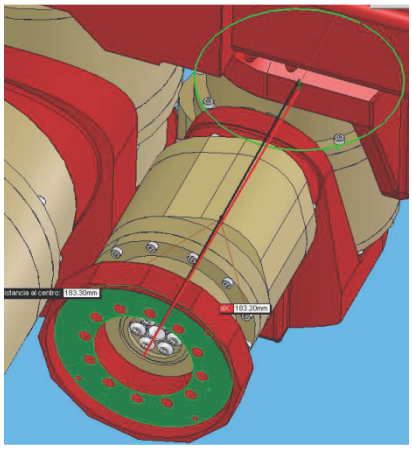
A2 = 495mm



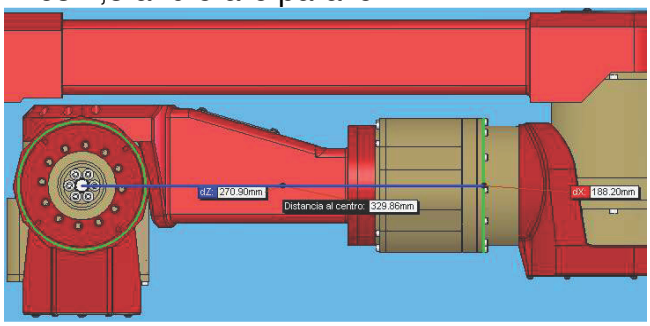
D4=372mm



D6 = 183.2mm



Axes 1, 4, and 6 are aligned (dy=0).
Axes 2,3 and 5 are parallel.



According to the figures, one DH table of the Modular Robot Arm could be:

robotnik (6 axis, RRRRRR)
grav = [0.00 0.00 9.81] standard D&H parameters

alpha	A	theta	D		R/P
1.570800	0.000000	0.000000	0.291100	R	(std)
0.000000	0.495000	0.000000	0.000000	R	(std)
-1.570800	0.000000	0.000000	0.000000	R	(std)
1.570800	0.000000	0.000000	0.372000	R	(std)
-1.570800	0.000000	0.000000	0.000000	R	(std)
0.000000	0.000000	0.000000	0.393300	R	(std)

1.1.1 Denavit Hartenberg parameters for the standard arm models

Standard DH – Parameter Table

Axis	theta	D	alpha	A	q0
1	0	d1	pi/2	0	0
2	0	0	0	a2	pi/2
3	0	0	-pi/2	0	-pi/2
4	0	d4	pi/2	0	
5	0	0	- pi/2	0	0
6	0	d6	0	0	0

Standard Arm – Med Range – 1200

Variable	Value
d1	240
a2	540
d4	415
d6	171.5+110

Standard Arm – Short Range - 850

Variable	Value
d1	240+53.1(*)
a2	395
d4	271.53
d6	171.5 + 220

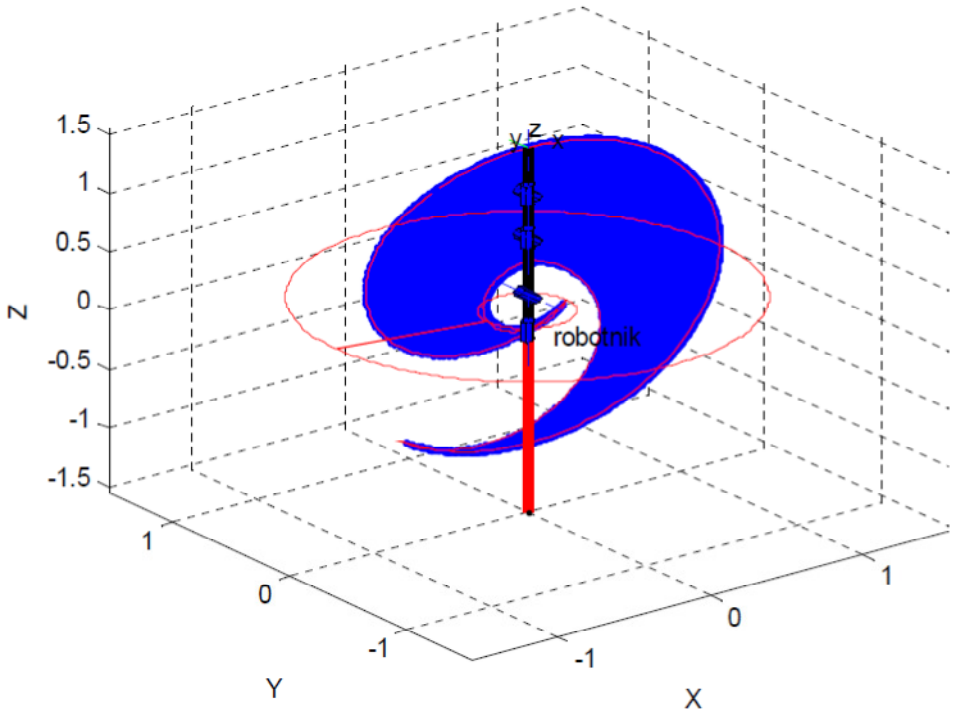
(*) including base flange

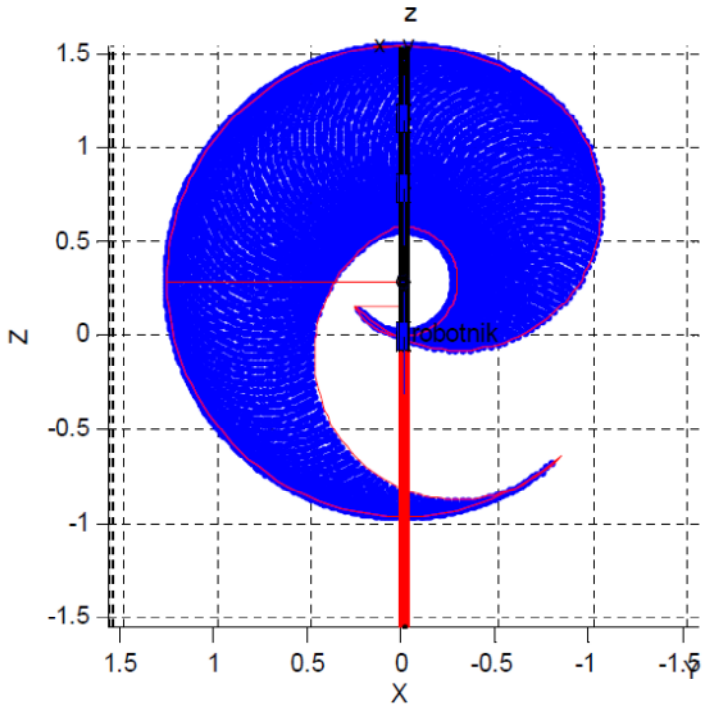
1.2 Robot Workspace

The following figure shows the robot workspace according to the joint range limits. The robot joints have been limited to avoid self collision of the arm (mechanical restriction, e.g. joint 5 can create a collision if the angle is greather than 90º), but also to reduce the amount of kinematic redundant solutions (e.g. joints 4 and 6 can rotate infinitely, but this would only increase the number of solutions increasing complexity.

Joint	*Min*	*Max*
q1	*-180º*	*180º*
q2	-126 º	126 º
q3	*-100º*	100 º
q4	*-180º*	*180º*
q5	*-90 º*	*90 º*
q6	*-180º*	*180º*

For some applications (e.g. screwing with the 6th joint) it makes sense to overpass the programmed ranges. If this is performed, it is important to notice that the internal cabling imposes also a restriction in the number of turns.





ANEXO III. Informe reflejado en el archivo 'Chequeo.txt'.

Año::2013 Mes::10 Día::10
Hora: 13:18:30

Init String: ESD:0,1000

Modulo 1::
Found module 1 with SerialNo 40487
Posición: 0

Modulo 2::
Found module 2 with SerialNo 40489
Posición: 0

Modulo 3::
Found module 3 with SerialNo 40583
Posición: 0

Modulo 4::
Found module 4 with SerialNo 40477
Posición: 0

Modulo 5::
Found module 5 with SerialNo 40476
Posición: 0

Modulo 6::
Found module 6 with SerialNo 40415
Posición: 0

13:18:30

ANEXO IV. Hojas de características.

Robotnik

BRAZO MODULAR

Producto

El **brazo modular** de Robotnik es un brazo robot formado a partir de los servoaccionamientos modulares *PowerCube* de Schunk. Estos accionamientos integran motorreductor, etapa de potencia y controlador, de forma que el brazo resultante no requiere de un armario de control externo. En su lugar, la conexión externa del brazo se reduce a la alimentación a 24 VDC y la de comunicación a través de CAN bus.

Los módulos trabajan como controladores distribuidos. El controlador maestro (PC de control) es el encargado de generar la secuencia del programa y el envío de referencias a cada uno de los ejes del sistema de articulaciones.
El control de corriente, velocidad y de posición, tiene lugar en cada uno de los módulos, así como las operaciones de supervisión de temperatura y control de parada. También existe la posibilidad de cerrar los lazos de control (corrriente, velocidad y posición) en el controlador externo.

Los elementos y brazo se ajustan a los requerimientos de cada aplicación, pudiéndose elegir un brazo convencional o rediseñar los elementos de unión con los parámetros de cada aplicación específica.

Este tipo de brazo, que integra la etapa de potencia y control, es el más adecuado para su instalación en plataformas móviles o androides. Entre las ventajas que ofrece este tipo de brazos, destacamos:

- Alimentación a 24 VDC. No requiere cargar con un pesado inversor.
- Control a través de bus CAN. A diferencia de un brazo robot industrial, solamente se requiere un ordenador con una tarjeta de comunicación CAN, en lugar de un armario de control.
- Las dimensiones de los enlaces, así como el número de grados de libertad, se hacen de acuerdo con las especificaciones del cliente.

El brazo modular suministrado integra:

- Brazo robot integrado por *PowerCubes* y los elementos de enlace entre los módulos.
- Elemento de ensamblaje con el chásis o la base donde se vaya a instalar.
- Elemento de ensamblaje con el efector final.
- Conectorización y cableado.
- Tarjeta de comunicación CAN.
- Software de control para plataformas Windows y Linux. Incluye algoritmos de control de colisiones, control de trayectorias, cinemática directa e inversa.

Robotnik Automation S.L.L C/ Berní y Catalá, 53 bajo izq. 46019 Valencia

Tel.: 96 338 38 35 / Fax: 96 338 35 80 www.robotnik.es - info@robotnik.es

Robotnik

BRAZO MODULAR

Aplicaciones

- Manipulación móvil.
- Robótica humanoide.
- Robótica médica.
- Robótica de servicio.
- Control de fuerza (con sensor de 6 g.d.l.).

Especificaciones técnicas

Mecánicas y Electrónicas

Dimensiones	1300 mm alcance con brazo extendido (7 g.d.l)
Peso	19 Kg con una configuración de 6 g.d.l.
Capacidad de carga	9 Kg en la base del efector final con 6 g.d.l.
Grados de libertad	Desde 4 a 7 g.d.l + efector final
Material	Aluminio
Motorización	Servoaccionamientos *PowerCube* Motorreductor y etapas de potencia y control integradas Frenos electromagnéticos en todos los ejes
Alimentación	20 A @ 24 VDC Potencia máxima 6 g.d.l. 1 A @ 24 VDC Control 6 g.d.l.
Rango de movimiento	Depende de la configuración y el nº de grados de libertad Para 6 g.d.l. : (+-) 190º, 105º, 180º, 195º, 107º, 360º
Velocidad de movimiento	57 grados / s en la base 300 grados / s rotación muñeca Resto de ejes depende de la configuración y g.d.l
Precisión	Repetitividad posicional de 0.5 mm
Modularidad	Sistema ampliamente escalable - Permite montar combinaciones desde 4 a 6 g.d.l. - Permite la conexón de cualquier tipo de sensorización - Permite montar uno o más brazos en una misma plataforma

Control

Comunicación	CAN
Controlador	Control de corriente, velocidad y posición en cada uno de los módulos Posibilidad de cerrar los lazos de control desde un controlador externo
Software	Librerías de control para Windows y Linux Implementado el control de colisiones, control de trayectorias y cinemática directa e inversa

Robotnik Automation S.L.L C/ Berní y Catalá, 53 bajo izq. 46019 Valencia

Tel.: 96 338 38 35 / Fax: 96 338 35 80 www.robotnik.es - info@robotnik.es